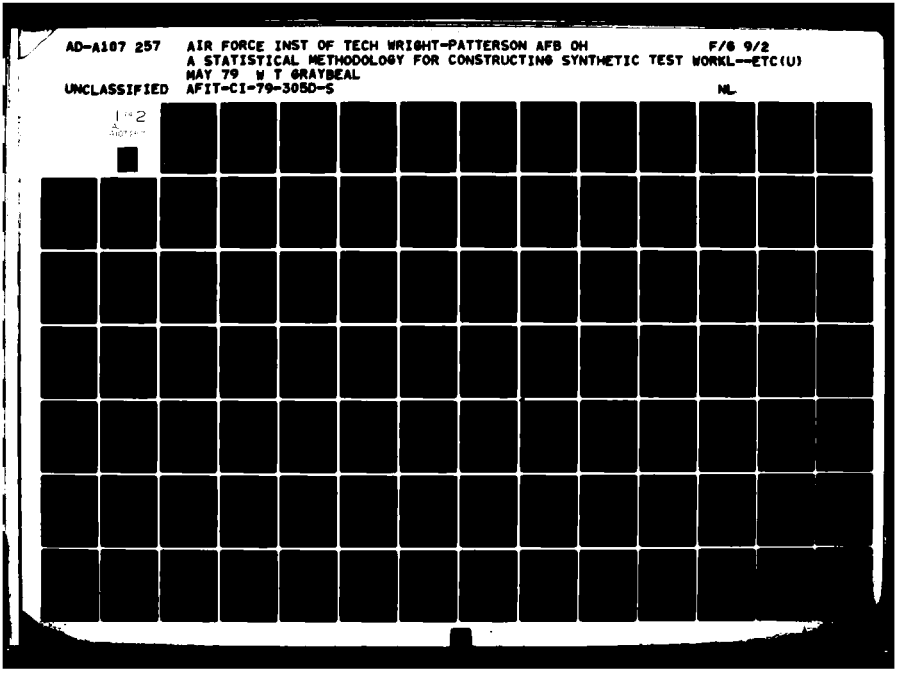
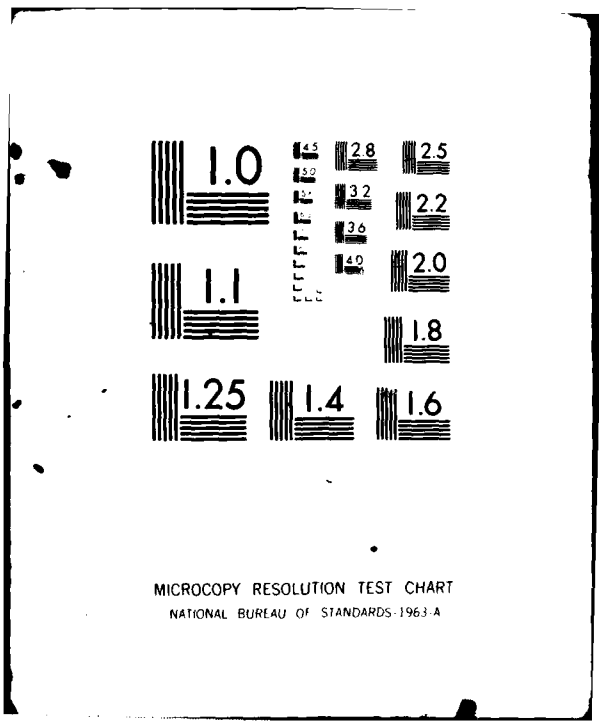
AD-A107 257 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH F/G 9/2
A STATISTICAL METHODOLOGY FOR CONSTRUCTING SYNTHETIC TEST WORKL--ETC(U)
MAY 79 W T GRAYBEAL
UNCLASSIFIED AFIT-CI-79-305D-S NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1 REPORT NUMBER: 79-305D-S

2 GOVT ACCESSION NO: AD-A107257

3 RECIPIENT'S CATALOG NUMBER:

4 TITLE (and Subtitle): A Statistical Methodology for Constructing Synthetic Test Workloads.

5 TYPE OF REPORT & PERIOD COVERED: THESIS/DISSERTATION

6 PERFORMING ORG. REPORT NUMBER:

7 AUTHOR(s): Wayne Thomas Graybeal

8 CONTRACT OR GRANT NUMBER(s):

9 PERFORMING ORGANIZATION NAME AND ADDRESS: AFIT STUDENT AT: Texas A&M Univ

10 PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Doctoral thesis,

11 CONTROLLING OFFICE NAME AND ADDRESS: AFIT/NR, WPAFB OH 45433

12 REPORT DATE: May 1979

13 NUMBER OF PAGES: ~~163~~ 178

14 MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15 SECURITY CLASS. (of this report): UNCLASS

15a DECLASSIFICATION DOWNGRADING SCHEDULE

16 DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT-CI-79-305D-S

17 DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

27 OCT 1981

Fredric C. Lynch

FREDRIC C. LYNCH, Major, USAF
Director of Public Affairs
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

18 SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17

19 KEY WORDS (Continue on reverse side if necessary and identify by block number)

20 ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD A107257

DTIC FILE COPY

81 11 03 040

ABSTRACT

A Statistical Methodology for

Constructing Synthetic Test Workloads (May 1979)

Wayne Thomas Graybeal, B.S., University of Oklahoma

M.A., University of Arizona

Chairman of Advisory Committee: Dr. Udo W. Pooch

Computer performance measurement and evaluation (CPME) studies are conducted for the purpose of sizing and selecting a new system (selection studies); during the design phase of either a new system or a hardware/software modification to an existing system to assess the impact of the new system/modification (performance projection studies); or to assess and improve the level of performance of an existing system (performance monitoring studies). Nearly all performance measures used are related to the workload being processed by the system. There is the need for a workload which emulates the actual workload, yet executes in less time and does not compromise the adequacy of the measurements. Such a workload is called a drive or test workload.

A statistical methodology is proposed to aid in the construction of a test workload. The major elements of this methodology are

(a) selecting the workload subset by constructing an overall workload profile and then choosing a period which exhibits characteristics pertinent to the evaluation study,

(b) choosing a set of descriptor variables which is detailed enough to represent the demand placed upon the major system resources, but is not so detailed as to complicate later stages of the analysis,

(c) collecting data reflecting the values of the descriptor variables for the worksteps in the selected subset,

(d) scaling the resource demand matrix so that each descriptor has mean 0 and variance 1,

(e) applying principal components analysis to the scaled resource demand matrix and retaining only those components needed to explain the major part of the variability in the data,

(f) clustering the transformed resource demand vectors in the principal components space using a non-hierarchical clustering algorithm with a weighted Euclidean distance measure,

(g) designing synthetic jobs for each of the isolated clusters using regression analysis to obtain predictor equations for the parameter settings,

(h) forming a synthetic job mix by combining a sufficient number of copies of the various synthetic jobs with appropriate parameter settings and the desired arrival time of each, and

(i) validating the generated synthetic job mix by executing it on the system being studied, comparing its resource demand characteristics with those of the real subset, and adjusting the parameter settings as necessary.

A detailed case study of the workload processed by the Amdahl 470/V6 at Texas A&M University is presented illustrating many of the proposed techniques. Suggestions for further work are included.

A STATISTICAL METHODOLOGY FOR CONSTRUCTING SYNTHETIC TEST WORKLOADS

A Dissertation

by

WAYNE THOMAS GRAYBEAL

Submitted to the Graduate College of
Texas A&M University
in partial fulfillment of the requirement for the degree of

DOCTOR OF PHILOSOPHY

May 1979

Major Subject: Computing Science

ATCH 2

A STATISTICAL METHODOLOGY FOR CONSTRUCTING SYNTHETIC TEST WORKLOADS

A Dissertation

by

WAYNE THOMAS GRAYBEAL

Submitted to the Graduate College of
Texas A&M University
in partial fulfillment of the requirement for the degree of

DOCTOR OF PHILOSOPHY

May 1979

Major Subject: Computing Science

A STATISTICAL METHODOLOGY FOR CONSTRUCTING SYNTHETIC TEST WORKLOADS

A Dissertation

by

WAYNE THOMAS GRAYBEAL

Approved as to style and content by:

(Chairman of Committee)

(Member)

(Member)

(Member)

(Head of Department)

May 1979

ABSTRACT

A Statistical Methodology for
Constructing Synthetic Test Workloads (May 1979)
Wayne Thomas Graybeal, B.S., University of Oklahoma
M.A., University of Arizona
Chairman of Advisory Committee: Dr. Udo W. Pooch

Computer performance measurement and evaluation (CPME) studies are conducted for the purpose of sizing and selecting a new system (selection studies); during the design phase of either a new system or a hardware/software modification to an existing system to assess the impact of the new system/modification (performance projection studies); or to assess and improve the level of performance of an existing system (performance monitoring studies). Nearly all performance measures used are related to the workload being processed by the system. There is the need for a workload which emulates the actual workload, yet executes in less time and does not compromise the adequacy of the measurements. Such a workload is called a drive or test workload.

A statistical methodology is proposed to aid in the construction of a test workload. The major elements of this methodology are

(a) selecting the workload subset by constructing an overall workload profile and then choosing a period which exhibits characteristics pertinent to the evaluation study,

(b) choosing a set of descriptor variables which is detailed enough to represent the demand placed upon the major system resources, but is not so detailed as to complicate later stages of the analysis,

(c) collecting data reflecting the values of the descriptor variables for the worksteps in the selected subset,

(d) scaling the resource demand matrix so that each descriptor has mean 0 and variance 1,

(e) applying principal components analysis to the scaled resource demand matrix and retaining only those components needed to explain the major part of the variability in the data,

(f) clustering the transformed resource demand vectors in the principal components space using a non-hierarchical clustering algorithm with a weighted Euclidean distance measure,

(g) designing synthetic jobs for each of the isolated clusters using regression analysis to obtain predictor equations for the parameter settings,

(h) forming a synthetic job mix by combining a sufficient number of copies of the various synthetic jobs with appropriate parameter settings and the desired arrival time of each, and

(i) validating the generated synthetic job mix by executing it on the system being studied, comparing its resource demand characteristics with those of the real subset, and adjusting the parameter settings as necessary.

A detailed case study of the workload processed by the Amdahl 470/V6 at Texas A&M University is presented illustrating many of the proposed techniques. Suggestions for further work are included.

ACKNOWLEDGEMENT

The author's graduate studies were funded by the United States Air Force through a program administered by the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, Ohio.

Deepest gratitude is extended to Dr. Udo W. Pooch, Chairman of the author's Graduate Advisory Committee. Dr. Pooch provided immeasurable assistance in defining and conducting this research.

Appreciation is also extended to Dr. Dan D. Drew, Computing Science, Dr. Glen N. Williams, Computing Science, Dr. Larry J. Ringer, Statistics, and Dr. Morgan O. Reynolds, members of the Advisory Committee.

Successful completion of this research would not have been possible without access to "real-world" resource utilization data. The assistance of Dr. Dick B. Simmons, Director of the Data Processing Center, Texas A&M University, and his staff in this regard is gratefully acknowledged.

A very special note of gratitude is extended to the author's wife, Annie. Without her extraordinary patience and diligence in typing the manuscript, the project could not have been completed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Background

The development of the electronic digital computer, begun in the late 1940's and continuing until the present time, has had a dramatic effect on nearly every field of human endeavor. Rapid advances in both hardware and software have exceeded even the most ambitious projections. With each advance in hardware and/or software came another level of complexity. This led to the ultra-fast, highly sophisticated systems of today in which the synergistic effects of their combined hardware and software subsystems can yield performance which is surprising even to the system designer. It has been suggested [78] that these systems are too complicated for the problems they are intended to solve, and that their complexity makes them inherently inefficient. The degree of truth in these suggestions may be debated, however it is apparent that the computer has evolved into one of the most complicated systems yet devised by man.

The Communications of the Association for Computing Machinery is used as a pattern for format and style.

Computer Performance Measurement and Evaluation (CPME) is a term coined to refer to a loosely-defined branch of computer science. It has evolved to satisfy the need for understanding and predicting the performance of computer systems. As the name implies, there are two different aspects to the study of the performance of a computer system. The first is measurement, the act of ascertaining the extent of the performance. The second is evaluation, the act of examining or judging the value of performance [29]. This field is not a new one [20], however, recent technical advances in hardware and a rethinking of the problem have led to a broadening of scope. While the early researchers were concerned with only the performance of the hardware [64], the performance of a given computer system has been realized to be a function of the total hardware/software package. Thus, such seemingly unrelated areas as program behavior [85], computational complexity [6] and software engineering [38] have been recognized as having an impact on the performance of computer systems.

1.2 Types of Evaluation Studies

The development, acquisition and maintenance of a computer system is an expensive proposition. Unfortunately, an efficient and an effective system appears to be the exception rather than the rule [56]. Thus, there is a continuing interest, both on the part of management as well as system analysts, in the understanding and in the improving of the performance of computer systems.

Lucas [64] classified evaluation studies by the reasons for which

they are conducted. Selection evaluation studies are conducted for the purpose of sizing and selecting a new system. This type of evaluation assumes that the relative performance in accomplishing a certain task is a factor in choosing one system over another system. Performance projection studies, on the other hand, are conducted during the design phase of either a new system or a hardware/software modification to an existing system. The aim of such a study is to assess the impact that certain features of the new system or subsystem will have on the system's performance. Such an evaluation is handicapped in most cases by the lack of a prototype. The results obtained are therefore largely theoretical and subject to validation once the system or subsystem design is implemented. The third type of evaluation is termed performance monitoring. This type of study has as its aim the assessment and improvement of the level of performance of current systems. Results of this type of evaluation can be used to "tune" a system, thus attaining a higher level of efficiency; to establish a profile of system activity in order to apply priority algorithms and establish billing procedures; or to forecast the impact of a proposed change in either the system or the workload.

There appears to be a degree of commonality in both purpose and technique in the classifications proposed by Lucas[64]. On the other hand, a more meaningful classification might be one proposed by Svobodova [88]. A study which is conducted to assess the performance of one system relative to another is called a comparative evaluation.

A study that is conducted to evaluate the system's performance relative to system parameters and/or system workload is termed an analytical evaluation.

1.3 Evaluation Techniques

Three general techniques have emerged in the evaluation of computer systems: analytical, simulation, and empirical [39]. The technique to use is affected by such factors as why the study is being conducted, the level of detail needed in the study, and the availability of the system being studied.

Analytical techniques are characterized by the representation of the system in the form of a mathematical model, and the solution techniques using ordinary mathematical means. Probably the most common mathematical model of a computer system is that of a queuing system [10,17,19,21,27,37,77]. In this representation, the system or subsystem being studied is considered as a service facility. Jobs or tasks are considered as customers arriving to the service facility requiring some quantity of service [40,58]. There are a number of disadvantages to using analytical techniques in an evaluation study. First, a mathematical model which is detailed enough to accurately represent today's highly complex computer system is likely to be mathematically intractable. Second, in an effort to make the model solvable, the researcher may be required to make a number of assumptions. For example, if a queuing model is used, it is common to assume that the interarrival times are independent and that the system has achieved a stochastic balance (steady state) [40,88]. The validity

of such assumptions can certainly be questioned in many studies. A third disadvantage is that even if the system is accurately represented and the assumptions deemed valid, the researcher has the problem of estimating system parameters. For these reasons, analytical techniques have found little utility in full-scale performance evaluation studies. They have, however, been used in studies involving subsystems or particular aspects of a system's behavior such as CPU scheduling [57,68], and the management of I/O channels [34,84].

The second general technique used in evaluation studies is simulation. In this technique, the structure of the system is reflected in a computer program. The behavior of the system under particular conditions can then be studied by varying the parameters of the simulator. This technique avoids the problem of intractability encountered in analytic methods, and generally does not require the researcher to make as many assumptions. There are, however, problems with this technique as well. If a high degree of detail is required in the system model, the simulator can become quite expensive to develop and to use. Furthermore, to be a useful tool, the simulator must be validated. That is, it must be demonstrated that the simulator behaves in the same manner as the real system when presented with identical conditions. Often this aspect of the simulation study is neglected [39], which leads to questionable interpretation of any results. There are many examples in the literature [60,61] of full scale system simulations.

The third general category of evaluation techniques involves studies made through the observation of some real system (empirical

analysis). This generally entails the collection and analysis of data reflecting the system's performance. Much data is collected through accounting logs and other means at every computer installation. It has only been recently that a serious attempt has been made at analyzing this data. Empirical techniques, aside from their utility in conducting separate evaluation studies, also provide a means of validating results obtained from an analytical or simulation study [39]. Problems encountered in using empirical techniques include the unavailability of the system and the degradation of system performance because of the monitoring process.

1.4 Performance Measures

In the past, the relative performance capability of a computer system was judged by such hardware characteristics as CPU cycle time, memory access time and the time needed to execute particular operations (i.e. add) [64,68]. It was thought that the shorter these times were, the more "powerful" the system was and hence the higher its performance rating. In later years, especially with multiprogrammed systems, it has become apparent that, although important, these measures are generally inadequate in characterizing the performance of a given system. Many other "performance measures" have been developed and are considered to be more useful in assessing performance. Some of the more popular of these measures are detailed below. For a more complete list, see Svobodova [88].

One of the more common measures of the performance of a computer system is throughput. Throughput is defined to be the amount of useful work completed per unit time when executing a given workload [9,56,88].

Since throughput is generally used in comparative evaluation studies, a related measure, relative throughput, has been developed. The relative throughput is defined as the ratio of the elapsed time required to process a given workload on one system versus the time required to process the same identical workload on another system [29,41,88]. Still another related measure is the throughput rate, defined to be the average number of task completions per unit time [75].

With the advent of multiprogrammed systems, a number of measures were developed to assess the performance of these systems relative to monoprogrammed systems. One of these is the Elapsed Time Multiprogramming Factor (ETMF) which is defined [82,88] as the ratio of the turnaround time of a job in a multiprogrammed environment to the turnaround time when it is the only job in the system. Another related measure is the gain factor [88] which is the total system time needed to execute a set of jobs in a multiprogrammed environment to the total system time needed to execute the same set of jobs serially. Still another measure related to multiprogramming is the internal delay time [88], which is the ratio of processing time of a job in a multiprogramming environment to the time required when it is the only job in the system.

Other advances in software and hardware necessitated more measures of a system's performance. For example, virtual memory systems necessitated a measure of the behavior of page and segment replacement rules. Page (segment) fault rate [23,88] is the most frequently used measure of this performance.

1.5 Means of Measurement

An empirical performance evaluation study requires that data be collected on the system activity. There are a number of ways this data can be collected. The simplest way [75] is through a simple observation of the system utilizing the system console and the behavior of I/O units as an indication of system performance. The type of information which could be gained through this type of observation would appear to be severely limited. Another source of information on the behavior of the system is from the system accounting logs . These logs can be used to obtain information on resource utilization at a job or job-step level. A third source is utilizing a monitor. There are three general types of monitors available: hardware, software and hybrid.

A hardware monitor [15,75] is logically and physically distinct from the system being monitored. System activity is routed to the monitor through a series of probes. For instance, the period of time a processor spends in the WAIT state could be monitored by installing a probe on the line leading to the WAIT light on the system console. Hardware monitors, though they can be used to measure essentially any event, are limited in that they cannot give an indication as to the cause of the event.

An alternative to the hardware monitor is the software monitor. Software monitors are programs which reside on the system being monitored. There are two general types of software monitors. The first, the interrupt-intercept monitor [75] is activated whenever an event which causes an interrupt occurs. Rather than control being passed

directly to the interrupt handler, it is instead routed to the monitor which records the system state and then passes control to the appropriate interrupt handler. The second type, the sampling monitor, is activated at certain time intervals, at which time it records the system state. Regardless of which type of software monitor is used, there is a serious drawback. That is since the monitor is resident in the host system, it competes for system resources along with normal jobs. Thus, the use of a software monitor can degrade system performance through the introduction of additional system overhead. This degradation has been termed the "artifact" of using a software monitor. This artifact can be a serious problem in evaluation studies, since the results obtained on system activity are biased to some degree.

In an effort to minimize the disadvantages of pure hardware and software monitors, the hybrid monitor has been developed. The hybrid monitor is essentially a combination of the two previous approaches. A minicomputer is normally attached as an "intelligent" terminal to the host computer. Hardware probes are used to detect event occurrences, just as in the pure hardware approach. In addition, the hybrid monitor has the ability to interrupt the host system and cause status information to be sent to it. Thus, a hybrid monitor can link event occurrences to their causes, which pure hardware monitors cannot. Further, since the required software support within the host system is limited, the software monitor artifact is reduced. This approach to monitoring appears to be the most promising.

1.6 Influence of Workload on System Performance

Nearly all of the performance measures mentioned earlier related the performance of a system to a particular workload. It has long been recognized [13] that the choice of the workload will have a major impact on the observed performance. For example, if one of the aspects of system performance that is being studied is the percent of channel utilization, an I/O bound workload would provide entirely different results from that of a compute-bound workload.

The workload (jobload) of a computer system is defined [75] as the set of all programs, data, and commands that are submitted to the system for subsequent execution. Since a workload has such a dramatic effect on the performance of a given computer system, the problem of how to represent or characterize the workload has arisen in practically every computer system evaluation study undertaken [32]. In many cases, workload characterization is the hardest technical problem to solve for the investigator [32]. There are many reasons for this, the chief one being the nonrecurrent nature of a computer workload. That is, if a system is handling a repetitive workload in which the same set of requests are made cyclically, then the workload characterization problem could be solved simply by examining the set of requests made in one cycle. Unfortunately, in most cases the workload is not repetitive, hence no general model can be developed.

Executing the entire job profile on each potential computer system that is to be evaluated can be expensive and time consuming . Thus, there is the need for a workload which emulates the actual workload, yet

executes in less time and does not compromise the adequacy of the measurements. Such a workload is called a drive or test workload.

The form of the test workload depends upon the techniques used in the evaluation study. If empirical studies of the system are made, the test workload will consist of an executable job stream. When analytical models of the system are used, the test workload could be represented in the form of interarrival and service distributions [23]. A simulation study would require an abbreviated job description in a form compatible with the simulator that uses this workload.

1.7 Properties of Test Workloads

Regardless of the form of the test workload, there are a number of properties which the test workload should possess to enhance its usefulness in an evaluation study. Ferrari [32] lists eight such properties. Some of the more important of these properties are given below.

Representativeness. The most important characteristic of a test workload is that it be representative of the actual workload. A test workload is representative if the system's measured performance when executing the test workload approximates the system's measured performance when executing the actual workload. This definition implies the existence of a distance function or metric by which it is possible to measure the relative degree of representativeness between two candidate test workloads. Unfortunately, such a metric does not exist, since the degree of representativeness depends not only on the performance measures used, but also on the relative weights assigned to each measure. [31].

Reproducibility. Aside from being representative, a test workload must be reproducible. Comparative evaluations as defined earlier are designed to assess the relative performance of two or more systems. If the effects of the different performance capabilities of the systems are to be isolated, the same test workload must be executed on each system. If different test workloads are executed, any variation in the obtained performance measures could be due to either the test workload or the actual system differences. A second reason that system test workloads must be reproducible is that a replication of the basic evaluation experiment may be desirable. This repetition allows for greater credence in the results.

Flexibility. A flexible test workload is one that can be easily modified. A researcher may wish to modify the test workload for a number of reasons. First, the actual workload of a computer system is likely to change over time. If the test workload is to remain representative, it must be changed also. Second, in establishing the representativeness of a test workload, it may be necessary to iteratively adjust the characteristics of the test workload to realign the properties with those of the actual workload. The ease with which these changes can be made have an impact on the cost of the evaluation study, in terms of both time and expended resources.

Portability. A requirement in comparative evaluation studies is that the same workload be executed on a number of different systems. A test workload should be constructed so that it may be transported between systems with a minimum of effort. Severe modifications to a

test workload can lead to biased results such as those mentioned in the section on reproducibility.

1.8 Dissertation Topic

The purpose of this research is to investigate the development of test workloads. There does not appear to exist a unified, comprehensive methodology which would allow the systems analyst to produce a concise representative workload for use in system evaluation studies, although considerable work has been done in the characterization and representation of workloads. This research is designed to aid in the development of such a methodology.

Major goals of this research include:

(a) To investigate the characterization of a computer system workload at a gross system level (daily/hourly characteristics) to aid in the selection of interest periods in a performance evaluation study.

(b) To examine the input job stream at a job or job step level with the aim of characterizing the pattern of resource requests.

(c) To investigate the design of parameterized synthetic jobs, which can be used in the construction of test workloads.

(d) To attempt to establish a step-by step procedure which can be used by systems personnel in developing test workloads for use in evaluation studies.

(e) To examine the procedure of (d) with an eventual aim of automating as much of the procedure as appears feasible. Though full automation of the procedure is not a goal of this research, the anticipated difficulties in this automation process will be considered.

1.9 Dissertation Contents

This dissertation is organized according to the three phases involved in the development of test workloads. These phases are the

(a) representation of the real workload, the

(b) analysis of the real workload, and the

(c) construction of the test workload.

A literature review of the current state of the art is contained in Chapter II. The literature review surveys the attempts made in the past few years to solve the problem of test workload construction suitable for use in performance evaluation studies.

Chapter III addresses the problem of representing the real workload. Some considerations in selecting an appropriate subset of the real workload, choosing a set of descriptors to use in representing each workstep, and collecting data to obtain real workload values for the descriptors are outlined.

Chapter IV contains a description of various statistical techniques useful in analyzing the represented worksteps for similar resource demand patterns, and summarizing the often voluminous amounts of data in an accurate and succinct manner.

The actual construction of the test workload is described in Chapter V. Some considerations and techniques for designing synthetic jobs are outlined. Procedures for validating (verifying the accuracy) the synthetic job stream are also given.

Chapter VI consists of a detailed case study illustrating many of the techniques outlined in previous chapters. The test case is not carried to conclusion (i.e. a complete ready-to-run benchmark) due to

a need to limit the scope of the research. The details necessary to carry it to such a conclusion are outlined.

The research is examined with the aim of producing a description of a fully automated test workload generator in Chapter VII. The results of the research are summarized, the more important points originated in this research are delineated, and areas of future research are suggested.

CHAPTER II

LITERATURE SURVEY

2.1 Introduction

The workload of a computer system consists of all individual jobs and data that are processed by the system during a specified period of time [86]. One of the principal problems facing a researcher conducting a performance evaluation of a computer system is representing the system workload in a form compatible with the evaluation techniques employed. It was mentioned earlier that the test workload should be representative of the actual workload in order that valid performance measures can be obtained; reproducible to allow replication of the experiments and verification of questionable results; flexible to allow easy modification; and portable to minimize the effort required to transport the workload between systems. The criteria for a "good" test workload are, to some degree, opposing, requiring compromise on the part of the researcher.

Some of the factors influencing the development of a test workload are the selection of which jobs to include in the workload model, the characterization of jobs in the real workload, and the type of test workload to use. The approaches to this problem which have been taken in recent years will be surveyed in this chapter.

2.2 Selection of the Workload

In most evaluation studies it is not possible to execute the entire job profile on each potential computer system to be evaluated. Some workloads are non-recurrent in the sense that there is no readily discernible cyclic pattern of resource demands. Other workloads have an extremely long repetition cycle (i.e. one week), hence inclusion of the entire job profile for a given cycle would not be feasible. This requires that a subset of the actual workload be used in constructing a test workload.

Choosing which jobs are to be included in a test workload is not a well-defined task. Hellerman and Conroy [42] list three important criteria in selecting jobs. These are

(a) those jobs which are run most frequently,

(b) those jobs which account for most of the system time and resource use, and

(c) those jobs whose completion-time requirements are most critical to the system's mission. The identification of these jobs may be somewhat difficult.

Since the test workload will normally be constructed using only a subset of the actual workload, one approach to the selection of jobs is to use the techniques of statistical sampling [83]. Jobs are selected at random from the real workload for use in constructing the test workload. As with any sampling procedure, there is a risk of obtaining a non-representative sample, and thus constructing a test workload which does not resemble the actual workload.

Another approach to the selection of the workload is to divide the actual workload into classes based on job functions. Then a number of jobs could be selected from each class based on their proportion in the total mix [52]. This segregation of the actual workload into classes could be done manually, or automated through clustering algorithms.

Still another approach to the selection of the workload is to pick that period of activity which has the greatest influence on the problem being studied. For example, if the load on the system is being studied, an obvious workload to consider is the period of peak activity. It should be apparent that if this approach is taken, the test workload will not be representative of the entire workload. This may, however, not be a serious constraint on the validity of the study [13].

Once a subset of the workload is selected for inclusion into the workload model, data must be collected which reflect the characteristics of the jobs included. System accounting logs, such as IBM's System Management Facility [47], or trace facilities supplied with the system, such as IBM's Generalized Trace Facility [48], are ready sources of such data. If these facilities are not available, data must be collected with a monitor [75]. The first approach appears to be the more popular [4,46,83,91] since the data is available with essentially no required modification to the system. The second alternative has, however, also been used [10].

2.3 Characterization of the Workload

Before the characterization of a real workload can be made, a basic unit of work must be defined. In some evaluation studies, the unit of

work may be a transaction, while in others it may be a job or job-step. Evaluation studies involving a general purpose system may utilize both transactions and jobs. Different types of workloads are generally considered initially separate, however lend themselves to be combined to form a composite workload. In the remainder of this section, the job will be adopted as the basic unit of work. It should be recognized that similar considerations apply for transactions in a time-sharing/ interactive environment.

Jobs or job steps in a batch processing environment can be described by the type of processing required, or alternately by the demand they place upon system resources [86]. The first approach is termed the service demand representation, while the latter is termed the resource demand representation. When the service demand approach is used, some of the typical processing requirements might be compilation, sort-merge, or file updates [86]. The distribution of the total jobs among the different processing groups provides an indication of the nature of the workload. Since this description does not depend on the particular type of computer system (i.e. a program which requires compilation on one system will generally require compilation on another), it can be referred to as a system independent description. Independence of any given system means that it can be used in comparative evaluations involving heterogeneous systems. This characterization is highly desirable, particularly in selective evaluations in which a potential customer is attempting to decide which of two or more different vendor's equipment

will best satisfy those needs. The service demand representation is rarely feasible, since information on the processing requirements for each individual job in the work stream is difficult, if not impossible, to obtain.

An alternate characterization is obtained if the computer system is viewed as a collection of resources upon which the users (workload) place demands. Some of the resources common to many computer systems with corresponding demands include the processor (CPU time), I/O channels and devices (number of I/O activities), core memory (size of the region), and unit record devices (number of cards read or punched, number of lines printed) [86]. The demands for these resources can be considered as the characteristic variables of the real workload processed by the system. A job can be described by a set of these characteristics [1], and since the system only recognizes a job by its pattern of resource demands, two jobs with the same resource demands would be characterized and treated identically [86]. It should be noted that the resource demands of a given job will vary from one computer system to another. Thus, this characterization is <u>system dependent</u>, and should be only used in comparative evaluations involving homogeneous systems. Its main usefulness would appear to be in system improvement studies involving a single system.

Regardless of whether the resource demand or service demand approach is used to characterize the workload, a job can be represented by an n-tuple $v=(v_1, v_2, \ldots, v_n)$, where v_i represents the magnitude

of the demand for the $i^{\underline{th}}$ resource or service. Using this representation, a number of different approaches have emerged for selecting jobs and setting the levels of their demands for each of the resources or services. Ferrari [32] describes five such approaches.

The first approach involves constructing the probability distribution of the demand levels in the real workload. By sampling these distributions, the appropriate demand for each resource or service can be derived for each job included in the workload model. This method was used by Schwetman and Browne [81], and a simulation based on this technique was described by Rosen [78]. The sampling technique used would appear to affect the representativeness of a workload description produced by this method.

The second approach is to extract real jobs from the real workload by sampling the workload. The resource/service demands for these sampled jobs are used to characterize the jobs in the workload model. This method has been used by Shope, et al. [83] and Wood and Forman [91].

The third approach mentioned by Ferrari [32] is to partition the real workload into classes, each characterized by similar combinations of resource/service demand patterns. A suitable number of jobs can then be selected from each class, and the resource demands for these jobs used to characterize a job in the model. This approach has been used by Joslin [51], Hunt, et al. [46], Agrawala, et al. [4] and Mamrak and Amer [66].

The fourth general approach is to construct the joint probability distribution of the parameters in the real workload (i.e. resource/

service demands) and derive from this distribution the parameters of a set of jobs with the same distribution. Sreenivasan and Kleinman [86] proposed this method and applied it to the construction of a test workload for a batch-processing installation. The major drawback to this method would appear to be that if a number of parameters are present, the joint distribution becomes difficult to manage. The last technique considers a job as a Markov process in which the states of a job are specified in terms of the values or ranges of values of its resource/service demands. The state-transition probability matrices for the real workload are constructed and used to derive the sequences of values for each job's parameters. This approach was investigated by Lasseter, et al. [62] and a model using this approach was implemented by Lindsay [63]. A recent work [70] investigated the modelling of a job in which the states of the Markov model were the types of programs being executed during each succeeding job-step.

Regardless of which of the approaches is used, the result should be a workload model stated in parametric form. That is, the real workload will be represented as a series of jobs, each of which has a certain pattern of resource/service demands.

2.4 Types of Test Workloads

Test workloads can be classified as executable or non-executable [32] depending upon whether they are intended for use in empirical studies or analytical/simulation studies. Non-executable workloads are of two general types. The first type is the probabilistic or distributional workload. In this approach, the requests for resources or

services are represented as probability distributions. The real distribution of these demands is often approximated by such standard distributions as the geometric distribution, or the hyperexponential distribution [88]. The closeness of this fit is obviously a factor in the degree of representativeness achieved. This approach has been used by many researchers [8]. Since the thrust of this research is not toward analytical/simulation studies, no in-depth review of this representation was made.

Alternately, the test workload may be script of system demands based upon the observed requests of a previously executed workload. This approach is called a trace, since it traces out the set of demands of a previously executed workload. A trace may be so detailed as to indicate each individual machine instruction executed, or be a series of aggregate demands placed on combinations of system resources [22,67,90]. As pointed out by Svobodova [88], the representativeness of a system trace can be affected by the artifact introduced in the monitoring process. Again, since this approach is of use in analytical/simulation studies, no detailed review was undertaken.

The most obvious choice for a test workload in considering executable workloads, is to use the actual workload (or a subset of this workload) that the users submit. In this case, the test workload is known as a benchmark. Benchmarks reflect demands the users make on the system, and these user demands must be translated into demands for system resources. Natural workload models (benchmarks) have been investigated and used in a number of studies [14,51,79,87]. Their

use, however, has a number of drawbacks. These include

(a) the drive workload is not flexible since it is constructed from jobs with fixed characteristics,

(b) large amounts of data on auxiliary storage may need to be duplicated to enable the running of some real jobs and

(c) security or privacy considerations may prevent the use of some jobs [86]. These and other considerations have led to the investigation of alternate forms of executable workloads.

An instruction mix is an artificially constructed job which is composed of a precise mix of certain types of instructions. This type of test job was one of the first artificial models suggested for use in performance studies [32] and it is useful in comparing the relative throughput of processors [88]. The most common mix is the Gibson mix [35], although numerous others have been suggested [33,45]. There are some disadvantages to using instruction mixes which tend to severely restrict their applicability. These disadvantages include that their use is restricted to comparing systems with similar instruction sets, and that they fail to account for input-output [42].

Another model which has been used to represent jobs in a test workload is the standard job or kernel. These artificial jobs are constructed to exhibit a particular behavior, and thus they can not be easily modified. They are of use when a projection of the workload is needed. They have also been used to compare the relative performance of language translators. Many collections of standard jobs exist [44].

A type of artificially constructed executable workload which has

received considerable attention in recent years is the synthetic job. A synthetic job is a program which does not perform any "useful" computing, but when executed results in demands for system resources similar to the demands of the actual workload. Synthetic jobs are generally written in a high level language with parameters which allow for easy modification. These parameters normally allow the user to specify the size of the program, amount of CPU time used, number and types of files accessed, and the amount of I/O performed. Thus, similar to benchmarks, synthetic jobs represent the workload from the user's point of view. The use of synthetic jobs overcomes many of the disadvantages of benchmarks. Resource-oriented synthetic jobs are typified by the single adjustable job proposed by Buchholz [18]. Wood and Forman [91], and Sreenivasan and Kleinman [86] have successfully used the Buchholz job for constructing synthetic test workloads. Curnow and Wichmann [25] developed an Algol job to simulate many computational procedures. Oliver et al. [76] developed a series of five simple synthetic jobs and experimented with them in producing synthetic workloads. Functionally oriented synthetic jobs have been described by Joslin [51] and Lucas [65]. For interactive or time-sharing environments, the synthetic jobs are typically developed from scenarios that specify system-independent functional activities and include a designation of all actions, pauses and decisions made by the user. Work in developing approximately representative test workloads for interactive systems has been done by Karush [53], Nolan and Strauss [74], Wright and Burnette [92] and Crothers [24].

2.5 Validation of Test Workloads

The results of the workload characterization process described in section 2.3 should be a model of the workload. The test workload then can be generated from this workload model through suitable representation of the jobs making up the model. Since the aim of constructing a workload model is to obtain a representative test workload, the validity of the workload model should be assessed. Agrawala et al. [4] describe validation of workload models obtained through the clustering approach. They suggest that the workload model should be constructed from one set of data (the design set) and validated using a second set of data (the test set). The method of hypothesis testing [43] is suggested for use in such a validation process.

Ferrari [32] discusses validation of the test workload. The procedure suggested involves the execution of the test workload on the system being tested. The pattern of resource demands made by the test workload is then compared to the pattern made by the real workload. This validation procedure was followed by Schwetman and Browne [81] and Kernighan and Hamilton [55]. Ferrari [32] suggests that secondary performance indices, in addition to those primary indices which were used in constructing the test workload, be included for use in validation.

2.6 Summary

Various approaches used to generate representative test workloads have been surveyed in this chapter. It should be apparent from the number of approaches surveyed that there is no widespread committment

to any one single method. However, the approach which combines characterization using clustering analysis and implementation using synthetic jobs appears to be gaining favor as the most promising approach.

The characterization of workloads and its impact on computer performance studies is still not well understood. Those approaches which were surveyed have not passed the test of time. That is, in most cases, they are single examples of possibly useful procedures. Until they are used by other researchers, they remain simply suggestions on how one might proceed.

CHAPTER III

SELECTING THE WORKLOAD

3.1 Introduction

The workload of most general purpose computing systems is dynamic in the sense that it cannot be represented as a cyclic demand for resources with a manageable repetition period. Furthermore, the needs of a user community historically have tended to grow to match or exceed the capacity of the computing system. Though the type of computing done may not change dramatically, the number of users and their frequency of use will steadily increase over the life of a system [46].

The dynamic nature of the workload of a computer system basically reflects the diversity of users. For example, in a large university environment, jobs submitted to the computer system could include instructional jobs, research jobs, administrative jobs (i.e. grade reports), commercial jobs, and overhead jobs (i.e. billing, etc.). The resource demand characteristics of these various classes of jobs may be radically different. Instructional jobs are generally small jobs, which individually use minimal resources, but due to the sheer number of such jobs in the job mix, they become a significant part of the workload. Research jobs, on the other hand, are much larger jobs, hence individually account for a greater share of resource use than

instructional jobs. Administrative and overhead jobs may be difficult to categorize since they are run for many different reasons. It should generally be apparent that they make liberal use of input-output (I/O) facilities, since most billing/report functions require heavy access to stored data files.

Not only are the computational requirements of the various classes of jobs different, the frequency and timing of runs may be significantly different. The frequency and timing of instructional jobs are influenced by factors such as the beginning/end of the semester, when the particular programming assignment is due, and even the schedule of extra curricular activities. Research jobs are influenced by such things as project deadlines. Administrative/overhead jobs may be considered cyclic, since they are generally run at about the same time each month/semester. The pattern of submissions of all classes, except possibly overhead jobs can be affected by various operational strategies such as reduced rates at particular times of the day. The diverse nature of the workload (i.e. various types of jobs and different arrival patterns) hinders any characterization effort.

3.2 Constructing a Workload Model

A problem which has received a great deal of attention [3,4,5, 12,41,46,66] is the establishment of a model of a computer workload. This is, in a sense, an attempt to characterize the users of a computer system. Such a model is important from the viewpoint of management [46] since it aids in planning. That is, if the characteristics of

the user population are known, projections can be made, and orderly expansion or replacement of the present system may be facilitated. The approach taken to solve this problem has been statistical sampling. The workload of the computer system is observed over some period of time (i.e. a day, a month or a year). Random sampling of this collection of jobs is then performed to achieve a representative collection of jobs. This reduced collection is then analyzed to discern underlying characteristics. These underlying characteristics are then inferred to the population as a whole. There are a number of difficulties associated with such an approach. Among these are:

(a) A significant part of the workload may be in the form of a relatively small number of extremely large jobs. These may be excluded from the model merely by chance.

(b) The workload of a computer system is generally not static in time. That is, a workload model constructed using data from a particular period of time may not even resemble the workload present at some other time, particularly with respect to the relative proportion of various job classes represented in the model.

Even if a representative workload model can be constructed, there are other difficulties which minimize the usefulness of such a model to construct a test workload for use in a performance evaluation study. Some of these difficulties are detailed in the next two sections.

3.3 Environmental Impact on Resource Demands

The resource demand pattern for a given workstep (i.e. job,

transaction, or job step) is to some degree dependent upon its environment. There are the obvious differences in the timing of the resource demands. The recorded magnitudes of the demands may also vary significantly from one run to the next of the same program depending on the system loading at the time. This difference in resource usage (and hence in the amount charged) from one run to the next of the same program may be baffling and sometimes annoying to the user; it must also be considered when constructing test workloads for performance evaluation studies. That is, a workstep which is removed from its environment and included in a sample for an evaluation study may exhibit a decidedly different resource demand pattern in its new environment.

An example of a particular resource demand which is subject to environmental variations is the amount of central processor (CPU) time required to complete a task. In a recent study, Davies [26] reported significant variance in the recorded CPU time for the same compute-bound program run under differing degrees of system loading. It was found that the recorded CPU time tended to increase as the loading on the system got heavier. There are two sources cited for this variance in CPU times. The first, referred to as the "true variation" is due to differences between runs in such things as cache performance, paging behavior in virtual memory systems, and memory access speed if processing is overlapped with "cycle-stealing". The second cited source of variation is due to the non-repeatability of how the system charges time to user processes, system overhead, and the idle state.

The charging algorithm varies from system to system. IBM [47] recognizes the possibility of variations in CPU time between two runs of the same program, and attributes it to such factors as channel program retries, CPU architecture (core buffering), cycle stealing with integrated channels, queue searching (such as task switching) and pending interrupts. Although in many cases the variation of CPU time between two runs of the same program may be small, Davies [26] cites one instance in which two runs of the same program produced reported CPU times in the ratio of 1:2.

A second resource demand which is subject to large environmental variations is paging behavior. Paging activity is influenced by two factors: program construction, and system environment. A program which exhibits a high degree of locality of reference [23,85] will generally not incur as much paging activity as one which does not have this property. This generally will have no impact on selecting an appropriate workload since the structure of programs are not normally modified. It will, however, have an influence on the development of synthetic jobs which is considered later. The opportunity for environmental variations in the paging behavior of a program becomes clear when a particular paging strategy is considered. Consider, for example, the Least-Recently-Used (LRU) paging algorithm [23]. This is a demand-paging algorithm in that a page is only read into main memory when a reference to it is made. As long as main memory is not full, no replacement of pages is made. When physical memory is full, a strategy is employed to decide which of a program's pages are to be "rolled-out" to free space to read in the next referenced page. The LRU algorithm assumes

locality, and replaces that page which has not been referenced for the longest period of time. If and when that page is again referenced, it must be read back into main memory. Thus if a program is executing in an environment in which main memory is not fully used, it is likely to incur fewer page faults than if it is executing in a heavily loaded environment in which some of its pages have to be "rolled out" and then "rolled back in" upon the next reference to them. This, of course, can result in widely varying channel utilization rates as well as contributing to variations in CPU time and I/O time.

3.4 Selection of an Appropriate Workload Subset

A performance evaluation study is normally conducted for a specific purpose. Studies performed on a single system could involve such things as assessing the impact that various dispatching strategies have on the average turnaround time; assessing the effect that a different page replacement strategy would have on paging behavior; or assessing the impact that adding another increment of physical memory will have on the behavior of a virtual memory machine. Obviously, one would like the test workload to exhibit certain properties to enhance the study. For example, if the evaluation study involves assessing the relative behavior of two page replacement rules, and a test workload is employed which does not fully utilize physical memory, the results of the study are likely to be less than satisfactory. One must, then, match the test workload to the evaluation study to some degree.

Workload periods which are apt to be of interest in evaluation studies are likely to be extreme periods. That is, the analyst wishes

to examine the system when some feature of it is heavily loaded. This fact, along with the failure to account for environmental variations, would seem to severely limit the applicability of system workload models constructed by statistical sampling to performance evaluation studies. That is, it is highly unlikely that one could achieve a job mix which would "strain" the system in the desired manner through random sampling.

An alternative to constructing the system workload model is to select a period of system activity which exhibits the desired characteristics, and use that period for the evaluation study. Not only is the desired characteristic present, but the environment has been preserved, which would minimize the problem of environmental variations. There is some sacrifice made if this procedure is followed, however. That is that since there is no randomization in the assignment of worksteps to the test workload, one cannot expect that the workload is representative of the entire real workload. Hence, inferences of system behavior must be restricted to at most similar periods. This may not be too small a price to pay when compared with the alternatives.

Detection of abnormal system activity which may be of interest in performance evaluation studies is rather a trivial task. System accounting logs normally contain summary data on system activity at a level appropriate for such detection. That is, information on the number of jobs processed, memory utilization, CPU utilization, etc., on a per hour or per day basis is recorded for management information. This data can be summarized and displayed in the form of a gross system profile. Abnormal periods are usually apparent from such profiles.

Once the appropriate period is selected, it can be examined in more detail to insure that it does indeed possess the required characteristics. Although they did not use it for this purpose, Bear and Reeves [12] describe the system workload of the CDC CYBER 74 system at Wright-Patterson AFB, Ohio at a level which would be appropriate for selection of interest periods. A similar profile of the workload of the Amdahl 470/V6 at Texas A&M University is illustrated in the case study of Chapter VI.

3.5 Selecting Descriptors for the Worksteps

Once the period of interest has been selected, a set of descriptors by which real jobs can be represented must be selected. If system logs are used to obtain data on the real workload, this involves deciding which of the recorded items are essential to characterize each job's demand on the system. If a monitor is used to collect the data, this determination must be made prior to the installation of the monitor, to allow for the collection of appropriate data.

The number of descriptors used to characterize each job will, in general, have a dramatic impact on the representativeness of the generated test workload. That is, if too few descriptors are used, the analyst cannot hope to faithfully reproduce the system behavior. If too many decriptors are included, on the other hand, the analysis of the workload data is complicated.

Ideally, if the resource demand description of workload is applied, the workstep descriptors should completely specify the demands placed upon the various system resources. Some of the resources upon

which jobs place varying degrees of demand are

(a) central processing unit (CPU),

(b) I/O processors (channels),

(c) main memory, and

(d) peripheral devices.

The demand placed upon some of these resources are easier to characterize than others. For example, the demand placed upon the CPU is reflected in the elapsed time the CPU spends in the execution of the job. The demand placed upon main memory can be measured by the size of the maximum partition used by the job, the average partition size used, or if greater resolution is desired, the weighted sum of the various partition sizes and the time each such partition is utilized.

The characterization of the demands placed upon I/O channels and peripheral devices is somewhat more difficult. There are normally a myriad of peripheral devices attached to a general purpose computer system. It is highly unlikely that an evaluation study would require resolution to the extent of measuring the demands placed upon each individual device. A reasonable measure would appear to be the amounts of each particular type of I/O activity (i.e. tape, disk, unit record) done by the job. Most system accounting logs reflect a number of measures of I/O activity. These include I/O time, as well as the number of data transfers initiated on each channel. Though the number of data transfers is not a direct measure of channel activity since varying amounts of data can be transferred, it may be sufficient in many evaluation studies. For those requiring more precision, the system accounting data can be augmented with hardware monitor data reflecting

the average channel activity per data transfer [91].

Previous workload characterization efforts reflect a multitude of descriptor sets used to characterize the demands placed on system resources by individual jobs. Sreenivasan and Kleinman [86] used only two variables, CPU seconds and the total number of data transfers initiated (EXCP count). A third variable, amount of core utilized, was recognized as important but it was found that the vast majority of jobs required similar amounts of memory. For this reason, it was not included in the descriptor set. Hunt [46] used eight descriptors: cards read, lines printed, CPU time, Peripheral Processor Unit (PPU) time, central memory, tape drives charged, cost to user, and whether or not FORTRAN was used. Agrawala, et al. [3,4,5] used eight features: CPU time, executive request and control card charges, average number of 512-word core blocks used, number of job steps (programs) executed, wall clock time, I/O to FASTRAND or disk devices, I/O to tape, and I/O to high-speed drum devices. Mamrak and Amer [66] summarized the workload using seven features: CPU time, disc EXCPs, tape EXCPs, cards read, lines printed, DD cards, and core used in kilobytes, where an EXCP reflects an I/O request and a DD card (data and device specification card) reflects a file accessed.

As can be seen from the above examples, there is no widespread agreement as to what constitutes a valid feature set for use in characterizing the resource demands placed upon a computer system by a particular job. The problem appears to be somewhat dependent upon the particular system in use and involves considerable intuition on the

part of the analyst performing the study. A different set of descriptors, along with some justification for its use is described in the case study in Chapter VI.

3.6 Collecting Data for Construction of the Test Workload

Once the particular subset of the real workload applicable to the evaluation study is selected, and an appropriate feature set formulated, the data reflecting the feature values for each workstep in the subset must be gathered. It may be that data collection is done before the determination of an appropriate feature set or vice versa. These two phases are certainly complementary, since it will do no good to choose a feature which cannot be measured and it is a waste of resources to collect data on features which are not used in characterizing the workload.

If monitors are used to collect resource demand data, there is a need to be able to project when in the future the system workload may exhibit similar characteristics to the period selected for the study. That is, the period of interest for an evaluation study is normally selected using historical data in the form of a system profile. Unless the monitor was installed and data collected during that particular period, which is unlikely, a period in the future likely to exhibit the same characteristics must be projected, so the monitor can be "turned on" to collect the appropriate data. It then must be verified after the data is collected if in fact the projected period exhibited the desired characteristics. This problem, as well as the added cost of using a monitor, has caused most researchers attempting

to construct test workloads to use system accounting data.

The case for using system accounting data in characterizing the resources used by a particular job is strong. First, the user is charged according to the usage reflected in these logs. Thus, at least from the point of management, the logs reflect the usage of critical resources. Second, the data is collected already for other purposes. The system analyst then obtains the data essentially without cost, either "out-of-pocket" or in terms of additional overhead to the system. Techniques for the collection of data as well as the types of data available from the system logs at Texas A&M University are considered in the case study of Chapter VI.

3.7 Summary

The selection of an appropriate subset of the real workload to use in a system performance evaluation study is one of the first decisions the analyst attempting to construct a test workload must consider. The subset selected must exhibit certain characteristics to enhance the evaluation study being conducted. Previous approaches based upon statistical sampling are not likely to yield the desired workload, since they fail to account for environmental impacts on the resource demands, and may exclude certain key parts of the workload. An alternative is to construct a system profile using system accounting data, examine that profile to detect particular desired loading characteristics, and use all or a portion of the actual workload during that period in the study. The environment is thus preserved, and the analyst is assured that the particular behavior of the system being studied

will be induced by the workload.

Once the subset of the workload is selected, the demands placed upon the various system resources by individual jobs must be quantified through the selection of a set of descriptors. This choice involves achieving a balance between the resolution of the precise resources used and the computational complexity in the analysis phase. Collection of data reflecting the real workload values for the descriptor set selected is the last task associated with this initial phase. System accounting logs provide a readily available source of data, and normally provide adequate information on resource utilization.

CHAPTER IV

ANALYZING THE WORKLOAD

4.1 Introduction

The techniques outlined in Chapter III will produce a subset of the real workload which can be used to construct a test workload for use in a performance evaluation study. This subset is represented as a number of jobs, each described by some set of descriptors. The time of arrival to the system, possibly the originating location if operating in a distributed environment, and the appropriate values of the descriptors form a complete specification of each job's contribution to the overall workload of the system. A test workload can be generated by replacing each of the jobs on a one-to-one basis with synthetic jobs which exhibit the same or similar resource demands. This, however, can prove to be an extremely trying task if a large number of jobs are included in the workload subset. It requires designing a separate synthetic job to replace each real job in the subset. Previous studies [3,4,5,30,46,66,86] have shown that the workloads of computer systems tend to be composed of a relatively small number of job classes, with resource demands similar within each class. If such classes are present, the effort required in constructing the test workload will be considerably diminished, since one synthetic job can generally be

used to represent all members within a class.

Thus, there is the need for analysis of the real workload subset to detect and isolate those jobs which exhibit similar resource demand characteristics. This chapter will outline a statistical clustering methodology useful in such an analysis.

4.2 Scaling the Descriptor Values

In general, each job in the workload subset can be described as to its resource demand characteristics by a set of descriptor variables $X_1, X_2, \ldots, X_n$, where the value of X_j, j=1, 2, ..., n, represents the demand placed on the $j^{\underline{th}}$ resource. The magnitude of the demands are obviously expressed in different units. For example, CPU time may be expressed in seconds, while memory utilization may be expressed in kilobytes. There is no obvious comparison which can be made between the various units, thus this unit dependence must be removed before the analysis can proceed.

One approach to scaling the variables is to transform each of the X_j values by

$$X'_j = \frac{X_j - X_{j\,min}}{X_{j\,max} - X_{j\,min}}$$

where $X_{j\,min}$ is the minimum observed value of X_j in the workload subset and $X_{j\,max}$ is the maximum observed value of X_j in the workload subset. This transformation scales each of the varables to the same range, namely from 0 to 1. The mean of the scaled variables is

$\bar{X}'_j = \frac{\bar{X}_j - X_{j\,min}}{X_{j\,max} - X_{j\,min}}$, while the variance of the scaled variables is

$V(X_j') = \frac{1}{(X_{j\,max} - X_{j\,min})^2} V(X_j)$ where $\bar{X}_j$ is the mean and $V(X_j)$ is the variance of the original unscaled variables. This approach has been used in at least one study [66] to remove the dependence upon units from the workload data.

An alternate approach to scaling the variables was taken by Agrawala, et al. [3,4,5]. They defined X_j^{α} to be the α-tile of the observed values of X_j, and then linearly scaled using

$$X_j' = \frac{10(X_j - X_{j\,min})}{(X_j^{\alpha} - X_{j\,min})}.$$

This results in a feature space in which 100α% of the observed data points lie in the interval from 0 to 10. For example, if α is chosen as .98, 98% of the transformed values will lie in the interval from 0 to 10 [3,4,5]. The stated purpose behind such scaling is to produce an essentially uniform feature space which is not distorted by the presence of outliers. The mean of the $j^{\underline{th}}$ descriptor variable under this scaling is $\bar{X}_j' = \frac{10(\bar{X}_j - X_{j\,min})}{(X_j^{\alpha} - X_{j\,min})}$, while the variance is $V(X_j') = \frac{100}{(X_j^{\alpha} - X_{j\,min})^2} V(X_j)$.

A third approach to scaling is to standardize the variables. That is, to scale each of the j variables to mean 0, variance 1. This is accomplished by the relation

$X_j' = \frac{X_j - \bar{X}_j}{\sqrt{V(X_j)}}$. This transformation, although it has not been applied (at least as can be determined) in

workload studies, is probably the most common transformation in statistical studies.

There are a host of other transformations which could be applied to workload data to remove the unit dependence and provide commensurable ranges for the descriptor variables. There does not appear to be a clear cut choice among the transformations since they are computationally similar and all accomplish the basic purpose. Standardization provides some side benefits. That is, if this means of scaling is used, the scaled data measures the variability in terms of standard deviation units. Furthermore, since the original data is expressed in widely different units, this means of scaling is preferred as a prelude to a principal components analysis [2,72].

4.3 Accounting for Correlation Among Variables

As developed in the previous section, each job selected for use in a performance evaluation study can be represented by a vector $\vec{X} = (X_1, X_2, \ldots, X_n)$, where the value of X_j represents the magnitude of the demand for the $j^{\underline{th}}$ resource. If there are m jobs in the selected subset, the resource demand characteristics for the subset can be represented by an mxn matrix

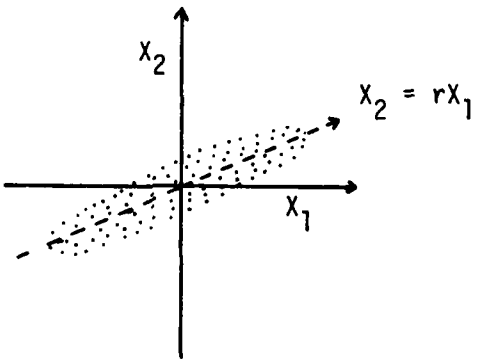
$$\vec{X} = \begin{bmatrix} X_{11}X_{12}\cdots X_{1n} \\ X_{21}X_{22}\cdots X_{2n} \\ \cdots\cdots \\ X_{m1}X_{m2}\cdots X_{mn} \end{bmatrix}$$

where the element X_{ij} represents the magnitude of the demand of the $i^{\underline{th}}$ job for the $j^{\underline{th}}$ resource.

The variables (descriptors) selected to measure the magnitude of the demands for resources for jobs in the selected subset will likely be correlated to some degree. That is, there is a degree of linear association among the variables. For example, it may be noted that jobs which print many lines of output have relatively large values of I/O time, or that jobs which incur a high degree of paging issue an inordinately large number of disc I/O requests.

The effect of intercorrelation among descriptor variables on the resource demand pattern of the workload subset can easily be visualized in two dimensions. Let X_1 and X_2 be two descriptor variables, which are correlated with a correlation coefficient $r>0$. If a scatter plot of the standardized values of X_1 and X_2 is constructed, an elliptical pattern oriented along the line $X_2 = rX_1$ will result, similar to that depicted in figure 4.1.

Fig. 4.1 The Effect of Correlated Variables



Intercorrelation among the descriptor variables will bias clustering results obtained when jobs are clustered by similar resource demands [16]. The effect is to provide a weighting for the common

characteristics reflected in the different variables. The severity of this bias is difficult to assess in general, since it is somewhat problem dependent. That is, it is related to the degree of intercorrelation, the distance metric used, and the weighting scheme supplied by the analyst.

It should be noted that high degrees of correlation do not, in general, indicate causal relationships, since there are many instances of totally unrelated phenomena which exhibit high correlation. However, if two highly correlated variables are included in the descriptor set, the biasing effect will be the same whether a causal relationship exists or not. This bias may not be undesirable, but it should be considered since it may help to explain seemingly contradictory results obtained in the clustering phase.

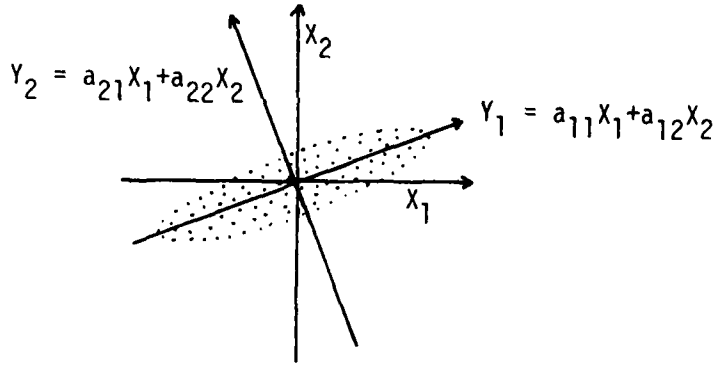
The problem of intercorrelation among descriptor variables is avoided if only uncorrelated variables are included in the descriptor set. This, however, is not feasible in most cases.

Given a set of n variables which are intercorrelated, it is possible to construct a set of n or fewer composite variables which are linear combinations of the original variables, are uncorrelated and which account for the variance in the data [7]. This can be accomplished by a method known as principal components [2,7,36,54,72,73,89].

Geometrically, the method of principal components involves a rotation of axes. Each of the resource demand variables $X_1, X_2, \ldots, X_n$ is represented by a coordinate axis from the origin $\vec{0} = (0, 0, \ldots, 0)$. These n axes form an n-dimensional space, with the $i^{\underline{th}}$ job represented by a point whose coordinates are $X_1 = X_{i1}, X_2 = X_{i2}, \ldots, X_n = X_{in}$.

In principal component analysis, the aim is to find a rotation of the axes so that the variable Y_1 represented by the first of the new axes has maximum variance. The variable Y_2 represented by the second of the new axes is uncorrelated with Y_1 and has maximum variance under this restriction. Similarly, the variable Y_k represented by the $k^{\underline{th}}$ new axis is uncorrelated with Y_1, Y_2, ..., Y_{k-1}, and has maximum variance under these restrictions [2]. The two variable case is illustrated in the following figure, where the "dots" represent the various jobs in the standardized resource demand descriptor space.

Fig. 4.2 Principal Components for n = 2.



Computationally, principal component analysis involves finding the eigenvalues of the correlation matrix of $\vec{X}$, choosing the eigenvectors corresponding to the nonzero eigenvalues orthonormal to each other, and postmultiplying the data matrix $\vec{X}$ by the matrix of eigenvectors. The details of this procedure are given in Appendix A.

The matrix $\vec{Y}$ which is produced by principal component analysis represents the scaled resource demand vectors of the workload subset

with relation to the orthogonal principal axes. The orthogonality insures that the new variables Y_1, Y_2, ..., Y_r are uncorrelated, hence clustering can proceed free of the biasing effect caused by the inter-correlations among the original variables. An additional advantage in possible reduction of the dimension of the feature space is gained by using this procedure, as will be discussed in the next section.

4.4 Reducing the Dimension of the Feature Space

If n resource descriptor variables X_1, X_2, ..., X_n are used to describe the demand placed on system resources, each job will be represented by a point in n-dimensional space. Prior to clustering jobs based upon similarity of resource demands, it may be advantageous to investigate the possibility of representing each job in a space of fewer dimensions. That is, it may be possible to depict the salient features of the resource demand patterns with $k<n$ descriptor variables. This is desirable from a computational standpoint, since the computational complexity of clustering is related to the number of descriptor variables as well as the number of data units (jobs in the workload subset).

The problem of reducing the dimension of the feature space has been examined in at least two workload characterization studies [3, 66], with somewhat contradictory results. Both studies approached the problem in much the same way. The scaled resource demand matrix was first input to a clustering algorithm with all variables present to achieve a "true" partition of the workload. A single resource descriptor was then removed, and the data matrix reclustered. This was

repeated, until all distinct sets of n-1 descriptors had been examined. The process then was applied to descriptor sets of size n-2, then n-3, and so on. The clustering performance for each set of descriptors was measured by examining the number of intercluster "migrations" as compared to the "true" partition. One study [66] reported promising experimental results using this procedure, while the other [3] downplayed its usefulness. This seeming contradiction of results is probably due to the differences in the two selected descriptor sets, and the different degrees of intercorrelations among those features reflected in the workload data. That is, if a descriptor variable which is highly correlated with another variable is removed from the descriptor set, its exclusion will likely cause fewer perturbations in the "true" partition than if a variable which is essentially uncorrelated with other variables is excluded. This again follows from the fact that correlated variables are, to some degree, reflecting the same characteristic of the workload.

Even if the above feature reduction algorithm proves useful in reducing the dimension of the feature space, it suffers from a fatal flaw. As previously stated, the aim of reducing the dimension of the feature space is to reduce the number of computations in the clustering stage of analysis. Since there is no a priori indication as to the relative worth of each descriptor in describing the "true" partition, one must cluster using all of the descriptors, and then iteratively reduce the dimension of the space. Thus, any computational advantage is lost. This problem is overcome to a certain degree if clustering is

applied to the principal component scores rather than the scaled variate scores.

Aside from the fact that its application produces uncorrelated variables, principal component analysis also is useful due to its maximum variance properties. The first principal component has the largest variance of any linear combination of the variables represented in the resource demand matrix; the second principal component has the largest variance of any linear combination orthogonal to the first principal component; the third principal component has the largest variance of any linear combination orthogonal to the first two, etc. This leads to a valuable property of principal components, namely that the best least squares fit of the original space of n dimensions in a space of k<n dimensions is achieved by using the first k principal components [7]. Thus, although to achieve a perfect fit, all of the principal components must be retained, if the analyst is satisfied with representing only a portion (say 95%) of the variability, a significant reduction in the dimensionality of the problem may be possible.

Information on the proportion of the total variability of the data matrix explained by the first k<n principal components is available without recourse to clustering. That is, it is a normal byproduct of principal component analysis. This measure is

$$P = \frac{\lambda_1 + \lambda_2 + \ldots + \lambda_k}{\lambda_1 + \lambda_2 + \ldots + \lambda_n}$$, where $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of the correlation matrix arranged in decreasing order.

Thus, by including enough components so that this ratio is at least as great as the minimum acceptable value, one can effectively reduce the dimension of the descriptor space and hence reduce the computational requirements in the clustering phase. It should be noted that this reduction of dimension is merely a reduction in presentation [72]. That is, measures on each of the original variables must still be taken since each may appear in the expression for a component variable. The aim of reducing the computational requirements in later phases is accomplished however.

4.5 Clustering Algorithms

Each job in the workload subset can be represented as an n-dimensional resource demand vector $\vec{X} = (X_1, X_2, \ldots, X_n)$ where the X_i are the magnitude of the demand for the $i^{\underline{th}}$ resource. Following scaling and principal component analysis, each job is represented as a k-dimensional vector $\vec{Y} = (Y_1, Y_2, \ldots, Y_k)$ in the principal components space. The next, and final, step in the analysis process is to cluster the jobs by similar resource demands, thus achieving a partition of the workload subset.

Prior to application of a clustering algorithm, the analyst must decide upon a measure of distance. That is, a measure must be selected which gives an indication of how "close" two jobs are with respect to their resource demands. A number of such distance measures are present in the literature. Probably the most commonly applied is the Euclidean measure given by

$$D(\vec{Y}_j, \vec{Y}_\ell) = \left[\sum_{i=1}^{k} (Y_{j,i} - Y_{\ell,i})^2\right]^{1/2},$$

where $\vec{Y}_j$ is the standardized resource demand vector for the $j^{\underline{th}}$ job in the principal components space, and $\vec{Y}_\ell$ is the similar vector for the $\ell^{\underline{th}}$ job.

Another consideration is the appropriate weight the analyst wishes to apply to each of the descriptors. That is, the analyst may wish to influence the clustering algorithm so that similarities in one dimension carry greater weight than similarities in another dimension. The weight W_i for the $i^{\underline{th}}$ descriptor is normally incorporated into the distance calculation as

$$D(\vec{Y}_j, \vec{Y}_\ell) = \left[\sum_{i=1}^{k} W_i (Y_{j,i} - Y_{\ell,i})^2 \right]^{1/2}.$$

Once the analyst has decided upon a distance measure and a weighting scheme, there are two general clustering schemes which may be used: hierarchical and non-hierarchical clustering [36].

The hierarchical scheme initially views the collection of m jobs as m separate clusters of one member each. A similarity measure is calculated between each pair of jobs, and those two jobs which are most similar are joined to form a cluster of two jobs. This cluster is generally represented by the average (centroid) vector of the two. This process is continued, with the two "closest" clusters joined at each step until the space is viewed as a single cluster with m elements. The analyst can halt the process at any time, thus achieving a partition with as many clusters as desired. This type of clustering scheme is typified by the algorithm proposed by Johnson [50].

Non-hierarchical clustering requires achieving an initial partition of the data set. There are a number of ways of achieving this initial

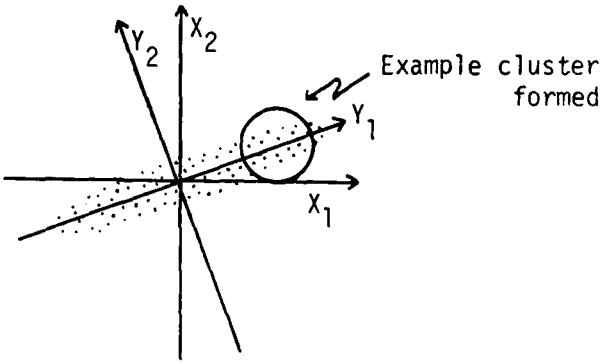
partition [7]. These include taking the first k jobs as cluster centroids, selecting some k jobs at random from the set as centroids, and taking a partition achieved by hierarchical clustering as the initial partition [7]. Once the initial partition is achieved, it is refined by comparing all jobs with the cluster centroids, and grouping those jobs with the "closest" cluster. The major differences in the various non-hierarchical schemes involve how and when the cluster centroids are updated and how many passes are made through the data. Non-hierarchical clustering schemes are typified by the k-means approach of MacQueen as described by Anderberg [7].

The decision as to which clustering algorithm to use is largely problem dependent. Hierarchical schemes generally provide more insight into the problem, since a wide range of partition sizes (number of clusters) can be examined with a single application of the algorithm. This, however, is counterbalanced by the fact that the non-hierarchical algorithms are more economical to use computationally, since they do not require the repeated calculation of similarity measures between each pair of data units [7]. Since the size of the workload subset is generally quite large (i.e. 750 jobs with 7 descriptor variables in one study [66]; 1342 jobs with 11 descriptor variables in another [3]) the insight gained through the use of hierarchical clustering is likely not worth the additional computational overhead incurred. Repeated application of a non-hierarchical clustering scheme such as one of the "nearest-centroid" algorithms detailed in Anderberg [7] will provide the needed insight at less cost in terms of computer time.

The bias caused by intercorrelated descriptor variables is

exposed through principal components analysis, however it is not eliminated. If an unweighted Euclidean distance measure is applied, hyperspherical clusters will be formed. Since expressing the resource demand vectors with respect to the principal components effects a simple rotation of the axes, clustering results using the unweighted Euclidean distance measure will be invariant under principal components analysis. That is, the same partition of the workload subset will result whether clustering with respect to the standardized variable scores or with respect to the principal component scores. This situation is illustrated for the two variable case in figure 4.3.

Fig. 4.3 Application of an Unweighted Euclidean Distance Measure



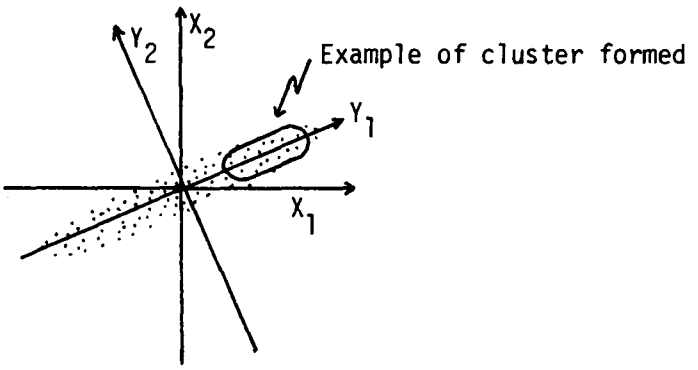
The bias caused by the correlation between the variables X_1 and X_2 is apparent in figure 4.3 by the "band" of data points in the cluster. Thus, the intracluster variance will be greater in the direction of correlation (Y_1) than in the direction orthogonal to it (Y_2). A weighting scheme is needed to equalize (or nearly so) the intracluster variations in both directions.

Application of a weighting function causes the formation of hyper-elliptic clusters [7], with the axes of the ellipsoids oriented along the variable axes. If a weighting scheme could be devised so that the intracluster variations in all directions are approximately the same, the biasing effect would essentially be neutralized.

If the data is subjected to principal components analysis, a measure of the variation along each of the component axes is available. That is, Var $(Y_j) = \lambda_j$. Intuitively, a weighting function W_i which is related to λ_i would appear desirable. Such a weighting scheme would weight the component variables in proportion to the variability that they "explain".

Suppose that the weighting function $W_i = 1/\lambda_i$ is applied to the component scores. This weighting function has precisely the same effect as standardizing the principal components and then clustering using an unweighted distance function. The effect of such a weighting scheme is illustrated in figure 4.4 for the two variable case, where it is assumed that $\lambda_1 > \lambda_2 > 1$.

Fig. 4.4 Effect of Improper Weighting

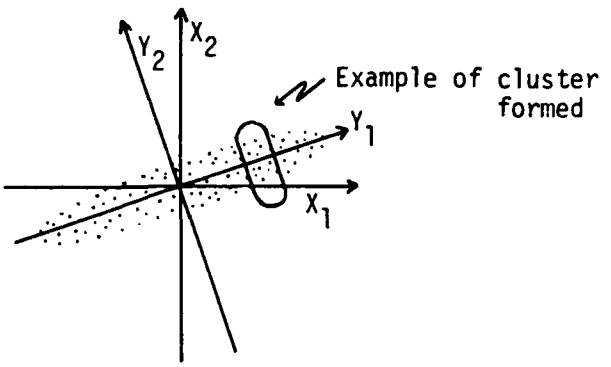


It can be seen from figure 4.4 that such a weighting scheme merely reinforces the bias rather than neutralizing it. That is, the intracluster variation in the direction of Y_1 is still greater than that in the direction of Y_2, even more so than if an unweighted distance measure were used. This type weighting then is not likely to improve the clustering results.

Suppose that a weighting function $W_i = \lambda_i$ were applied, where $\lambda_i > 1$. This should result in the formation of elliptic clusters whose major axes are orthogonal to those illustrated in figure 4.4. This weighting scheme is illustrated in figure 4.5 for the two variable case.

This weighting is seen to have the proper effect. That is, the intracluster variation in both directions are the same or nearly the same.

Fig 4.5 Effect of Proper Weighting



4.6 Summary

A statistical methodology has been proposed to aid in the analysis and summarization of the workload subset selected for use in an evaluation study. The major elements of this methodology are:

(a) Scaling of the data to commensurable ranges. A number of schemes are available to accomplish this, however, the standardization of all variables to mean 0, variance 1 offers some advantages.

(b) Applying principal components analysis to achieve uncorrelated variables and allow selection of some $k<n$ of the resource variables which account for the major part of the variance in the data.

(c) Applying a suitable clustering algorithm to associate "similar" jobs in the principal components space. A non-hierarchical scheme using a weighted distance metric appears the most promising.

An example of the application of this methodology to real workload data appears in the case study in Chapter VI.

CHAPTER V

CONSTRUCTING THE TEST WORKLOAD

5.1 Introduction

The output of the analysis phase will be a summarized form of the real workload subset. The jobs making up the subset are grouped according to similar resource demands. Each "cluster" of similar jobs is represented by the cluster centroid and a cluster membership list. Each of these clusters can be further analyzed by constructing distribution functions for each represented descriptor variable. This type of analysis would yield a workload model which could be used in analytic/simulation studies. Appropriate sampling techniques could be used to extract a test workload from such a model. Empirical studies, on the other hand, require that executable test workloads be constructed. Thus, the construction of distribution functions and sampling techniques will not yield a useful test workload for such studies.

A number of different types of executable test workloads were surveyed in Chapter II. These included benchmarks, instruction mixes, standard jobs, and synthetic jobs. Synthetic jobs offer advantages in in the areas of flexibility and portability over instruction mixes and standard jobs. They also avoid the security and privacy problems associated with using real jobs (benchmarks). A test workload composed of synthetic jobs, then, is likely to be the most useful form of an

executable test workload.

One of the primary criteria applied in assessing the usefulness of a test workload is how accurately it reflects the resource demands of the real workload which spawned it. A test workload which accurately reflects the characteristics of the real workload is said to be representative. Constructing a representative test workload using synthetic jobs requires careful design of the jobs making up the mix. Some of the techniques and procedures useful in designing synthetic jobs will be surveyed in this chapter. Most of the techniques surveyed are oriented toward test workloads constructed for a batch processing installation. Similar considerations apply to transactions in a time-sharing environment, however the general form of the model is different. The actions which must be emulated in an interactive session include user log-on, program creation, editing, program compilation, program execution, and user log-off. A model embodying such actions can more realistically be referred to as an interactive script [32] rather than a synthetic job.

5.2 General Considerations in the Design of Synthetic Jobs

A synthetic job is a parametric program in which the demands placed upon the various system resources are controlled by the values assumed by various input variables (parameters) [32]. This relationship to the actual resource utilization requires the programmer to approach the design of synthetic jobs from a different viewpoint than normal programming problems. Normal programming projects are usually undertaken for a particular reason. That is, the user wants the computer

to perform a particular task. The task to be performed is the overriding consideration in program development. There may be an attempt to minimize the resources used in an effort to hold down the cost of the project, but this is generally a secondary consideration. Synthetic jobs, on the other hand, are independent of the task which is performed. They are also independent of any input data or data files accessed by the real programs they are designed to emulate. The sole consideration in their design is that they use the same amount and types of resources that their real counterparts use. Thus, a somewhat arbitrary "compute loop" can be used to force the synthetic job to consume a particular amount of CPU time. I/O activity by real jobs can be emulated by having the synthetic job access arbitrary files of the required type (i.e. tape, disk, or card). These files can be "garbage files" expressly constructed for this purpose, or any other file to which the analyst has access. Thus, there is no unique synthetic job for each situation. A multitude of logically different programs can be forced to exhibit the same resource demand patterns with the proper choice and setting of parameters.

The degree of complexity of a synthetic job is generally determined by the level of detail used in characterizing the real workload. If a limited resource descriptor set is used, a relatively simple synthetic job will normally suffice. If, on the other hand, an expanded resource descriptor set is used which reflects more minute aspects of the real job's resource utilization, a more complex synthetic job will generally be required. Ferrari [32] illustrated this point with two examples.

The first example given by Ferrari [32] concerns construction of a test workload for a batch processing installation. Jobs in the workload were characterized by the descriptor pair (t_{cpu}, n_{io}). The first descriptor gives the CPU time required by the job while the second gives the number of I/O operations initiated by the job. Since the type of I/O is not specified, it can be assumed to be simple "reads" from cards and "writes" to a printer (or any other mode for that matter) in an arbitrary proportion. A synthetic job designed to emulate such jobs can be composed of a simple loop. I/O is performed a certain proportion of the iterations through the loop, and some arbitrary computation performed some other (or perhaps the same) proportion of the times through the loop. The loop is executed until the required number of I/O operations are performed and the proper amount of CPU time is accrued. An example of such a synthetic job and a situation in which this low level of detail is sufficient is given in the case study in Chapter VI.

More complex synthetic jobs are typified by the one developed and tested by Buchholz [18]. This job is designed to emulate a file processing action. There are three parameters used, which specify the number of master records read in, the number of detail (transaction) records processed, and the number of times the "compute" loop is executed. This job can be used to emulate the resource demands of jobs whose resource descriptor set is somewhat expanded over the earlier one described. An example of the use of such a synthetic job is also given in the case study of Chapter VI.

5.3 Parameterization of Synthetic Jobs

The parameters of a synthetic job allow the individual system resource demands to be easily modified. In general, greater flexibility requires more parameters, while simplicity and economy dictate that the number of such parameters be kept to a minimum. In the final analysis, it is the level of detail used in characterizing the real workload which determines the number of parameters to use. This required level of detail is in turn determined by the resolution necessary in the evaluation study. For example, consider a test workload composed of synthetic jobs where each synthetic job has parameters to specify memory size and total CPU processing time. This workload might be sufficient if the aim of the evaluation study is to determine the effects of altering main memory on CPU utilization. It would not provide the required resolution if the aim of the study is to determine the effects of differing amounts of I/O processing on CPU and I/O overlap. In fact this latter study would require at least one parameter to allow the ratio of CPU processing to I/O processing to be altered. It may also be necessary to include resource descriptor variables which specify the duration and relative timing of I/O requests. Thus, there is a three-way dependence among the performance measures observed in the study, the descriptor variables used to characterize jobs in the workload, and the synthetic job parameters used to control the demands placed on various system resources.

More formally, suppose that a test workload W_t is constructed for use in an evaluation study in which the ℓ performance variables V_1, V_2, ..., V_ℓ are to be observed. Suppose further that these performance

variables are functions of m system resources described by the descriptor variables $r_1, r_2, \ldots, r_m$, and that the values assumed by these descriptor variables are determined by n user parameters $p_1, p_2, \ldots, p_n$. The relations existing among the variables can be expressed as

$$V_1 = V_1(r_1, \ldots, r_m) = V_1 [r_1(p_1, \ldots, p_n), \ldots, r_m(p_1, \ldots, p_n)]$$
$$= V_1(p_1, \ldots, p_n)$$
$$V_2 = V_2(r_1, \ldots, r_m) = V_2 [r_1(p_1, \ldots, p_n), \ldots, r_m(p_1, \ldots, p_n)]$$
$$= V_2(p_1, \ldots, p_n)$$
$$\vdots$$
$$V_\ell = V_\ell(r_1, \ldots, r_m) = V_\ell [r_1(p_1, \ldots, p_n), \ldots, r_m(p_1, \ldots, p_n)]$$
$$= V_\ell(p_1, \ldots, p_n).$$

The relations can be summarized in more compact vector notation as $\vec{V} = \vec{V}(\vec{r}) = \vec{V}[\vec{r}(\vec{p})] = \vec{V}(\vec{p})$. Now, recognizing that the values assumed by the parameters $p_1, \ldots, p_n$ completely determine W_t, the composite relation $V_i = V_i(W_t)$, $i = 1, \ldots, \ell$, (or $\vec{V} = \vec{V}(W_t)$) results, where $W_t = W_t(p_1, \ldots, p_n)$.

One problem which must be solved in constructing W_t is determining the relationship which exists between the resource descriptor variables $r_1, \ldots, r_m$ and the synthetic job parameters $p_1, \ldots, p_n$. The parameters $p_1, \ldots, p_n$ can be assumed independent of one another, and in some cases they may bear a simple linear relationship to the r_i's. This relationship can be established by observing the r_i's for a few runs of the synthetic job with varying p_i's, and applying regression analysis [28]. The linear form of the relationships $r_i = r_i(p_1, \ldots, p_n)$, $i = 1, 2, \ldots, m$, allows inversion to give relationships of the form $p_j = p_j(r_1, \ldots, r_m)$, $j = 1, \ldots, n$. This assumes

$n \geq m$ and that the original system is non-singular. These latter relations can be used to determine the appropriate parameter settings to produce a given resource demand pattern.

Examples of the use of linear regression in establishing the relationships which exist between the resource descriptor variables $r_1, r_2, \ldots, r_m$ and the synthetic job parameters $p_1, p_2, \ldots, p_n$ are given in the case study of Chapter VI. It should be noted that the simple form of these relations does not suggest that similar simple relationships exist between the performance variables $V_1, V_2, \ldots, V_\ell$ and the resource descriptor variables $r_1, r_2, \ldots, r_m$. Establishing this relationship must be accomplished during the evaluation study itself.

5.4 *Controlling the Demand for System Resources*

A procedure for establishing the relationship between the resource descriptor variables $r_1, r_2, \ldots, r_m$ and the synthetic job parameters $p_1, p_2, \ldots, p_n$ was suggested in the previous section. This procedure assumes that parameters which are likely to affect the job's demand for a given resource have been established and incorporated into the design of the synthetic job. Some of the ways in which the demands placed upon system resources can be controlled are surveyed in this section.

One of the major system resources is main memory. The amount of main memory used by a given job is obviously related to the size of the program as well as the space needed for system routines supporting the job's execution. A job's main memory requirements can thus be

altered by modifying the size of arrays or by including routines which may never be called. A number of systems (i.e. IBM) enforce a policy known as "preallocation of resources" to preclude deadlock problems [23]. The maximum amount of main memory likely to be used by the job must be requested in advance of its initiation. If this requested amount is not sufficient to allow program execution, the job is terminated. The size of the region in main memory allocated to a particular program, if such a strategy is employed, can be either increased or decreased by altering the region request field in the job control statements.

Control of the amount of CPU processing time used by a program is possible by including a "compute-loop" control parameter. An arbitrary sequence of computations is performed iteratively until the desired CPU time is accrued. The required number of iterations through the loop can be controlled precisely through access to system timers [32]. It can alternately be established in advance through calibration experiments. The amount of processing time accrued by a particular job is related to factors other than simply the number of computations performed. The number of I/O activities initiated, for example, can have a significant impact on CPU time used.

Control of the I/O processing requirements of a job is more difficult than either main memory or CPU time. There are a multitude of different types of I/O. It may be necessary to control each of them, depending upon the resolution needed in the study. Unit record I/O (i.e. cards read, lines printed, and cards punched) is the easiest to control. The number of cards read is obviously a direct function

of the size of the program. It can be varied, within certain limits, by including or excluding comment and data cards. The number of lines printed (or cards punched) can be controlled through inclusion of a "print" (or "punch") loop. This loop is executed a sufficient number of times to produce the desired output. Tape and disk (or drum) I/O is controlled by creating files which are accessed using the proper mode. Records can be read, modified, and written under the control of a file processing loop. There is a potential problem in accurately reflecting the real workload's processing behavior. This results from the fact that in addition to controlling the number of I/O activities initiated, the size of the data block transferred each time must also be specified. Data on the real workload's resource demands is generally not available at the required level of detail from system accounting logs. It can be obtained by using a monitor, as was mentioned in Chapter III.

Another type of I/O activity which must be controlled in virtual memory systems is paging I/O. In a demand paging environment, blocks of data are transferred from auxiliary storage into main memory as required. If main memory is full, some "pages" may have to be recopied back to auxiliary storage to make room for the next "page" copied into main memory. Paging activity can be controlled to a certain extent by careful program development. Techniques useful in improving the locality of a program and thus decreasing its expected page fault rate are discussed by Spirn [85]. Paging activity is also highly environment dependent. Thus any significant control over paging activity will likely have to be exerted during the calibration/

validation phase when the entire test workload is available.

Direct control can be exerted over many of the system resources through inclusion of loop control parameters and proper job control statements. An example of the use of parameters to control the various system resources is included in the case study of Chapter VI.

5.5 The Design of Calibration Experiments

It is necessary once a synthetic job has been designed, to establish the relationship between the parameters of the synthetic job and the resource descriptor variables used to characterize jobs in the real workload. Such a process can be termed "calibrating" the synthetic job. The procedure proposed in Section 5.5 requires that the synthetic job be executed on the system for various parameter settings. The corresponding values of the descriptor variables are recorded for each run, and regression analysis used to establish the desired relationship. There are a number of unanswered questions associated with this procedure. These include how many runs of the synthetic program are necessary to establish an accurate relationship, what parameter settings should be used for each run, and how to account for the acknowledged environmental variations (see Chapter III) in the resource demands from one run to the next. The use of statistical experimental design techniques is proposed in this section to assist in answering these questions.

The magnitude of the demands placed on system resources by a given job can vary from one run to the next. Some of the demands most susceptible to these environmental differences are CPU processing time, I/O processing time, and data transfer over the channels handling

paging activity. This variation in resource demands can have a significant effect on relationships established through regression analysis. Indiscriminant running of the synthetic job will yield data in which it is impossible to separate the effect on the response variable due to this "chance" variation from that caused by the setting of various parameter levels.

Most of the parameters used in controlling the magnitude of the demands placed upon various system resources by a synthetic job can assume a wide range of values. For example, the number of times a "compute" loop is executed is constrained only to be a non-negative integer. Similar restrictions (or lack thereof) apply to other parameters. Failure to use a wide enough range of values for these parameters will yield a predictor equation which cannot be used in some cases. This is because it is almost never feasible to extrapolate using a regression equation [28].

Related to the setting of the parameter levels for each run of the synthetic job is the required number of runs. The synthetic job could be run a large number of times (say 100) with the parameters set at the same values. This obviously would yield a highly reliable relationship for that particular combination of settings. The validity of the relationship for some other combination of parameter settings would be highly suspect.

Problems similar to those outlined above are commonly encountered in other data analysis situations. A branch of statistics known as experimental design [43] has evolved to aid in the resolution of these problems. The methodology outlined for designing factorial experiments

[43] appears applicable to this problem.

A factorial experiment is one in which all levels of a given factor are combined with all levels of every other factor of the experiment [43]. Each of the synthetic job parameters to be varied can be considered as a factor in the calibration experiment. Levels for each factor can be established which are likely to cover the required range of resource demands. Each unique combination of factor levels can be thought of as a "treatment" to be applied. Treatments are assigned at random to each run of the job.

The use of statistical design techniques provides a number of advantages in calibration experiments. They include:

(a) The randomization of the treatment to run assignment minimizes the effect of chance environmental variations in resource demands.

(b) For a given number of factors and levels per factor, one can precisely calculate the number of runs necessary for a complete replication of the experiment. For example, if five factors are present, and each can assume two levels, $2^5 = 32$ runs are required. The analyst can reduce the number of runs by using fractional replications. This involves confounding some effects.

(c) The significance of the effects on the resource demands by the various parameters can be tested through an analysis of variance. Interaction effects can also be tested, although in some cases it is difficult to interpret such effects.

(d) Confidence limits can be established for the obtained regression coefficients.

It costs no more in most cases to conduct a carefully designed experiment than it does a poorly designed one. The use of statistical experimental design techniques can have a significant impact in the calibration phase. An application of these techniques is given in the case study of Chapter VI.

5.6 Validating the Test Workload

The calibration experiments discussed in the previous section can be used to establish predictor equations relating the synthetic job parameters to the resource descriptor variables. A synthetic job mix can then be constructed by including sufficient copies of each of the synthetic jobs with the appropriate parameter settings. It is necessary to execute this synthetic mix on the system being studied and to determine what degree of representativeness has been achieved. This process can be termed validation.

A number of authors [4,32,49,86] have emphasized the importance of validating test workloads. The general consensus seems to be that a test workload which has not been validated should not be used. The particular subset of the real workload which is used as a model in the design of a test workload is selected because it exhibits some characteristics pertinent to the evaluation study (i.e. heavy loading, high paging rate, etc.). If the test workload does not exhibit the same characteristics, the evaluation study can be severely hampered.

If the test workload does not accurately reflect the resource demands of the real workload subset, it is likely due to

(a) errors in recording the resource demands, either because the recording process was not accurate or because the resource demand

pattern was distorted (perhaps due to artifacts introduced by the monitoring process itself),

(b) errors introduced when the actual workload demands are reduced to probability distributions or clusters, or

(c) errors in computing the synthetic job parameters.

Errors of the first and second type are common to nearly all methods of generating test workloads. They can be precluded only by exercising extreme care in those stages of the construction process. Errors of the third type are unique to test workloads generated using synthetic jobs. Careful design of the calibration experiments should minimize the possibility of an error of this type occurring.

An obvious means of verifying the accuracy of the synthetic job parameters is to execute the test workload, record the demands placed upon the system resources, and then compare the resulting probability distributions of demand clusters with those produced by the real workload. A number of statistical tests (i.e. Chi-Square, Kolmogorov-Smirnov) are available for testing "goodness of fit". Errors of the first and second type mentioned above, however, could go undetected using this process. The monitoring process will likely introduce the same bias when the test workload is executed as it did during processing of the actual workload subset. The same analysis package will likely be used to summarize both the resource demands of the actual workload and those of the test workload. Thus, the same errors are apt to occur in both analyses.

The validation phase of test workload construction is probably the least understood phase. There are a number of reasons for this.

Many studies never progress this far, since it is the last phase of the process (although the calibration phase may be reentered if a non-representative test workload is produced). Secondly, to avoid distorting the demand characteristics of the test workload, it must be executed in isolation from other jobs on the system. This requires a dedicated system during that period of time, which is sometimes inconvenient and expensive.

5.7 Summary

A test workload can be constructed using synthetic jobs. The parameters to incorporate into the design of the synthetic jobs are determined by the resource descriptor variables used to characterize the real workload. These descriptor variables are in turn determined by the performance variables required by the evaluation study. Regression analysis can be used to establish the relationships between the synthetic job parameters and the resource descriptor variables. Statistical experimental design procedures can be applied to assist in the design of these calibration experiments. Following the design and calibration of the synthetic jobs, a synthetic mix can be constructed by including the appropriate number of copies of each synthetic job with the proper parameter settings. This test workload must be executed on the system, and its resource demands compared with those of the real workload. This latter process is termed validation.

CHAPTER VI

CASE STUDY

6.1 Introduction

A methodology for constructing a test workload suitable for use in a performance evaluation study has been developed in Chapters III, IV, and V. This chapter illustrates this methodology with a case study of the primary computing system at Texas A&M University.

A brief description of the present system configuration begins the study, followed by a description of the system workload in terms of gross workload characteristics. Succeeding sections illustrate the application of techniques to

(a) express the selected workload subset as a resource demand matrix;

(b) transform this demand matrix through suitable scaling and principal component analysis;

(c) summarize the workload subset using a clustering strategy;

(d) design synthetic jobs to replace the real jobs reflected in the selected workload subset.

This study is not directed toward measuring any particular aspect of the system's behavior. Rather, its aim is to demonstrate a procedure by which a drive workload can be constructed. For this reason,

there is a degree of arbitrariness in some aspects of the study, particularly in the workload subset which was selected. The selected subset does not exhibit any particularly outstanding feature; it was selected more or less at random. In an actual performance evaluation study, considerable care must be taken in selecting a workload subset which provides an appropriate environment for the study.

6.2 System Description

The Texas A&M University Computer Network is a centralized network with the Amdahl 470/V6 at its hub. Access through remote job entry (RJE) is possible from a number of locations throughout Texas, including Amarillo, Austin, Brenham, Galveston, Prairie View, Stephenville, Temple, Tyler, Texarkana, and Waco. In addition, four remote computing centers are dispersed about the main campus of Texas A&M. The Data Processing Center (DPC), which operates the network, acts as a centralized data processing facility, providing data processing services in support of the academic, research, and administrative functions of the university.

The Amdahl 470/V6, which was installed in late 1975, is the central computer. It is supplemented by various mini/micro computers which assist in data reduction and provide an opportunity for "hands-on" instruction. The 470/V6 is presently equipped with six megabytes of main memory, a sixteen kilobyte cache memory, and has a cycle speed of 32.5 nanoseconds. Sixteen data channels (0-F) are provided. These I/O processors are currently assigned as follows:

Channel 0 - Unit Record I/O

Channel 1 - 8 CALCOMP 3330 Mod I compatible disk drives

Channel 2 - Unit Record I/O

Channel 3 - 12 CALCOMP 3330 Mod II compatible disk drives

Channel 4 - COMTEN 3670 communications control module

Channel 5 - 12 CALCOMP 3340 compatible tape drives

Channel 6 - Alternate to channel 5

Channel 7 - Alternate to channel 3

Channel 8 - 80 IBM 3270 CRT terminals (IMS)

Channel 9 - Not utilized

Channel A - HASP pseudo devices (disk)

Channel B - 8 CALCOMP 3330 Mod I compatible disk drives

Channel C - Not utilized

Channel D - Not utilized

Channel E - Not utilized

Channel F - Not utilized

The system is presently operating under SVS Release 1.7, in a HASP 4.0 environment. SVS swaps virtual memory between the disk and real memory in 4096 byte segments (pages). TSO, the Time Sharing Option of IBM operating systems, provides a time sharing environment in which most functions available to the batch programmer are made available to the terminal user. Other software subsystems available include

(a) APL-SV - A time-sharing system provided by IBM which allows many terminal users concurrent access to the 470.

(b) IMS/VS - An IBM program product providing data base and

data communication facilities.

(c) SYSTEM 2000 - A general purpose data base management system developed by MRI Systems Corporation.

(d) MARK IV - A file management system developed by Informatics, Inc.

(e) PANAVALET - A program management and security system developed by Pansophics System, Inc.

(f) WYLBUR/370 - A text editing system developed at Stanford University.

A wide variety of language translators are provided. Those supported by the DPC include

(a) ASSEMBLER G - Assembly language,

(b) ASSEMBLER X - Assembly language,

(c) ASSIST - Fast student assembler,

(d) ANS COBOL (version 3) - Business oriented language,

(e) FORTRAN H (extended) - Scientifically oriented language,

(f) OS/VS COBOL - Business oriented language,

(g) PL/C - Fast PL/I compiler,

(h) PL/I Optimizing Compiler - General programming language,

(i) WATBOL - Fast COBOL compiler and

(j) WATFIV - Fast FORTRAN compiler.

In addition, language translators for ALGOL, SNOBOL, LISP, PASCAL, and RPG are available, but are not supported by the DPC. A large number of application packages are available, including GPSS, CSMP III, SSP, SAS 76, SPSS, and IMSL.

6.3 Workload Description

The workload of the Amdahl 470/V6 is composed of five general categories of worksteps, where in this case, "workstep" refers to an increment of the workload. This increment could be a job in a batch environment, or a session in a timesharing environment. These categories are:

(a) Teaching - student worksteps, and other worksteps run in direct support of teaching,

(b) Research - worksteps related to research projects,

(c) Administrative - worksteps run to support the everyday operation of the university,

(d) Commercial - worksteps run by non-university users,

(e) Overhead - billing programs and other worksteps run to support the operation of the DPC.

Although the proportion of the workload in each of these categories varies, during October/November 1978, the breakdown was Teaching - 58%, Research - 18%, Administrative - 6%, and Commercial/Overhead - 18%. It should be noted that these are proportions of the total number of worksteps processed rather than of total resource utilization.

For this study, the workload for the period January 1, 1978 to November 30, 1978 was examined. There were a total of 912,327 worksteps processed during this period, which accounted for 2944.66 hours of chargeable CPU time. The following relative frequency histograms show the distribution of the worksteps/ CPU time over the eleven month period.

Fig. 6.1 Relative Frequency Histogram for Worksteps Processed - Monthly

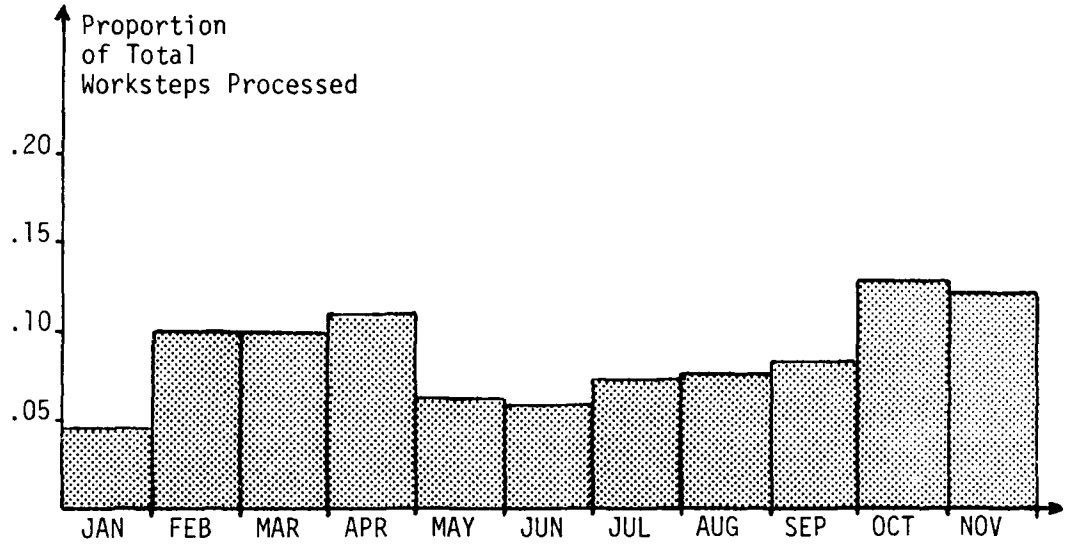
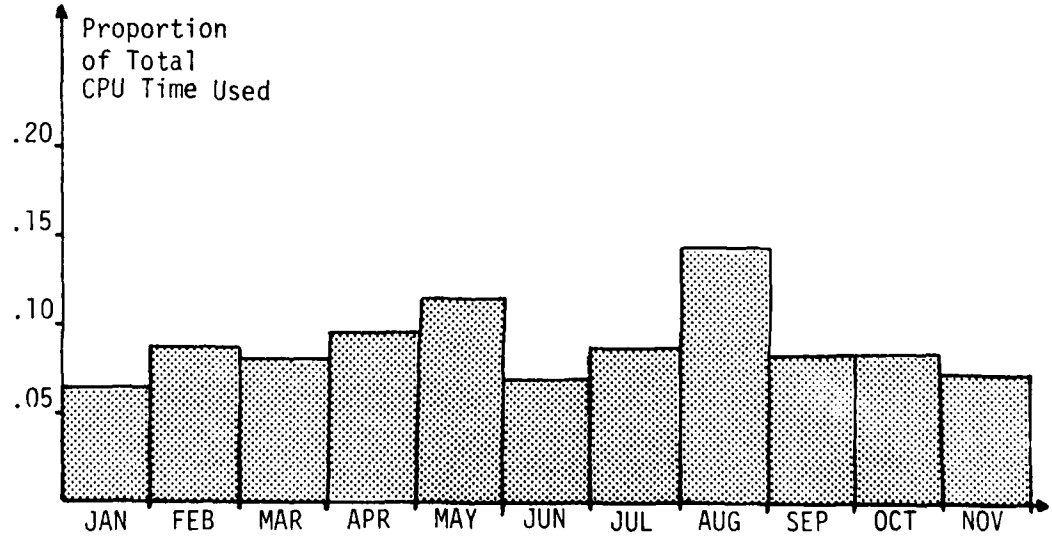


Fig. 6.2 Relative Frequency Histogram for CPU Time - Monthly



The structure of the histogram depicting the proportion of worksteps processed closely follows the academic terms. The spring semester began in late January and continued through mid May; the summer term ran from early June to mid August; and the fall term began in late August and ran into December. The histogram depicting the proportion of CPU time shows that the period of maximum utilization of the processor actually occurred during May and August, a time of relatively low student usage. This was caused by a heavy administrative workload during those two periods. Grade reports are processed in May accounting for that "hump"; both grade reports and normal end-of-the-fiscal-year processing account for the August "hump".

For this study, it was decided to examine a period which exhibited a balance in the types of worksteps processed. The period selected was a two week period, September 20 - October 3. This period should exhibit the desired balance, since it begins approximately one fourth of the way into the fall semester. Thus, the distortion caused by end-of-semester administrative processing is avoided. Furthermore, it is far enough into the semester so that student/research activity is relatively heavy.

The workload during the period of interest displayed a strong weekly trend. This is caused largely by the work week and operating hours of the various remote processing centers. There was a total of 46,730 worksteps processed during the two week period, which resulted in 127.03 hours of chargeable CPU time. The following relative frequency histograms depict the distribution throughout the week.

Fig. 6.3 Relative Frequency Histogram for Worksteps Processed - Weekly

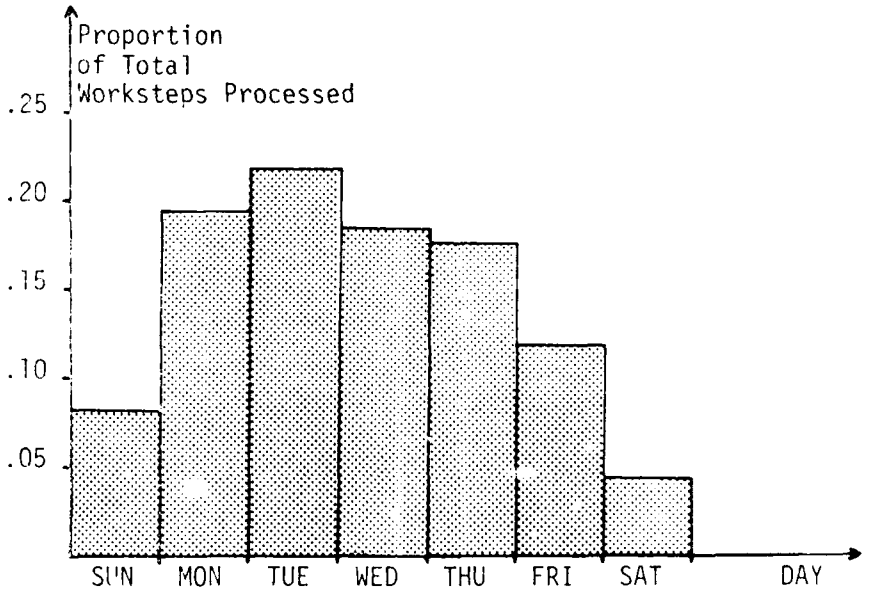
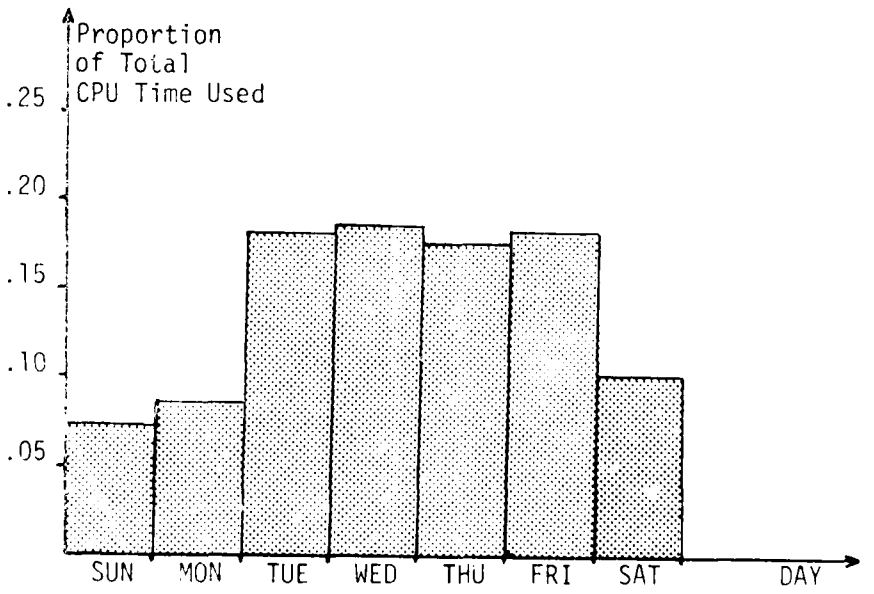
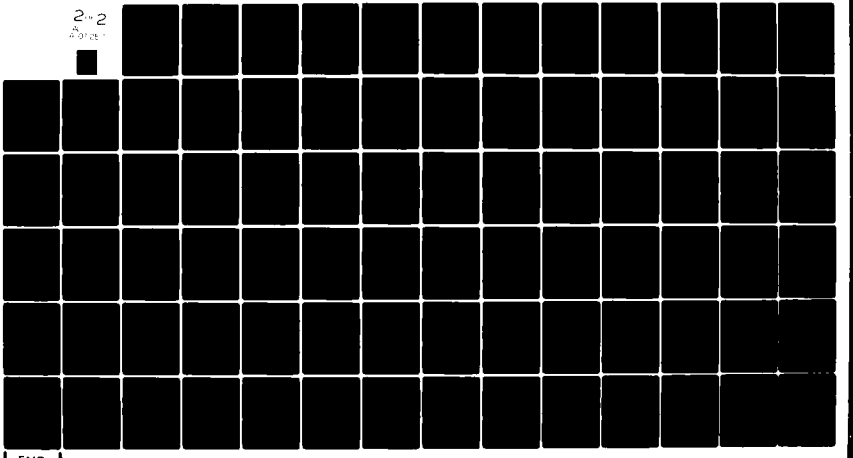


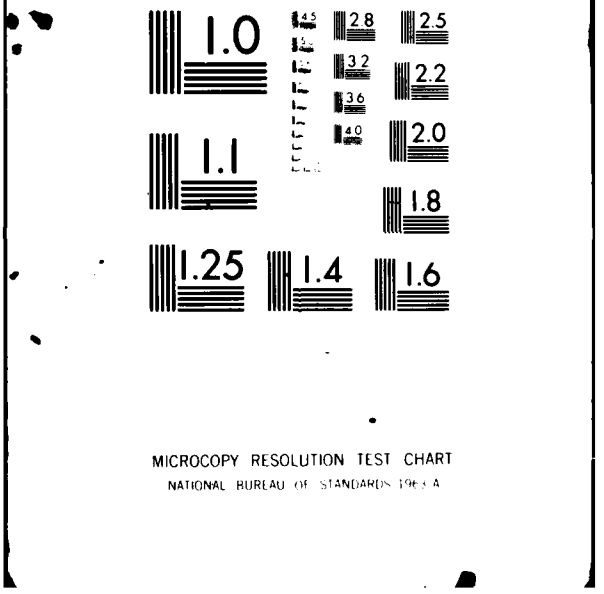
Fig. 6.4 Relative Frequency Histogram for CPU Time - Weekly



AD-A107 257 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH F/G 9/2
A STATISTICAL METHODOLOGY FOR CONSTRUCTING SYNTHETIC TEST WORKL--ETC(U)
MAY 79 W T GRAYBEAL
UNCLASSIFIED AFIT-CI-79-305D-S NL



END
DATE FILED
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

The seeming contradictions in the above histograms are caused by student jobs. A "happy hour" period is provided from 7:30 - 9:00 P.M. on Sundays; 12:30 - 1:00 P.M. and 8:30 - 10:00 P.M. on Mondays through Thursdays; and 12:30 - 1:00 P.M. on Fridays during which jobs using the student compilers are run without charge. Thus, the job counts during these periods are abnormally high. CPU utilization is not affected to the same degree, since these jobs are characteristically very minimal in terms of processing requirements.

Using this profile as a guide, a two hour period was selected as the workload subset for use in the remainder of the study. In an effort to keep the scope of the study reasonable, it was decided to restrict it to the batch portion of the system workload. It should be understood that to produce a realistic test workload, the interactive portion of the workload would also have to be considered. This analysis should parallel that of the batch workload, with the two types of workloads merged at the end to provide a composite test workload.

The two hour period form 9:00 - 11:00 A.M. on September 20, 1978 was selected, again to yield a balanced workload. This period avoids the influx of student jobs caused by "happy hour", and is contained within the normal workweek so that administrative/overhead jobs are represented. There were 338 jobs processed during this period, with 170 of them compiled using the in-core student compilers (Autobatch jobs) and 168 of them using the standard OS translators (Batch jobs). These two portions of the batch workload were analyzed separately due to the severe restriction in resource utilization placed upon the Autobatch jobs.

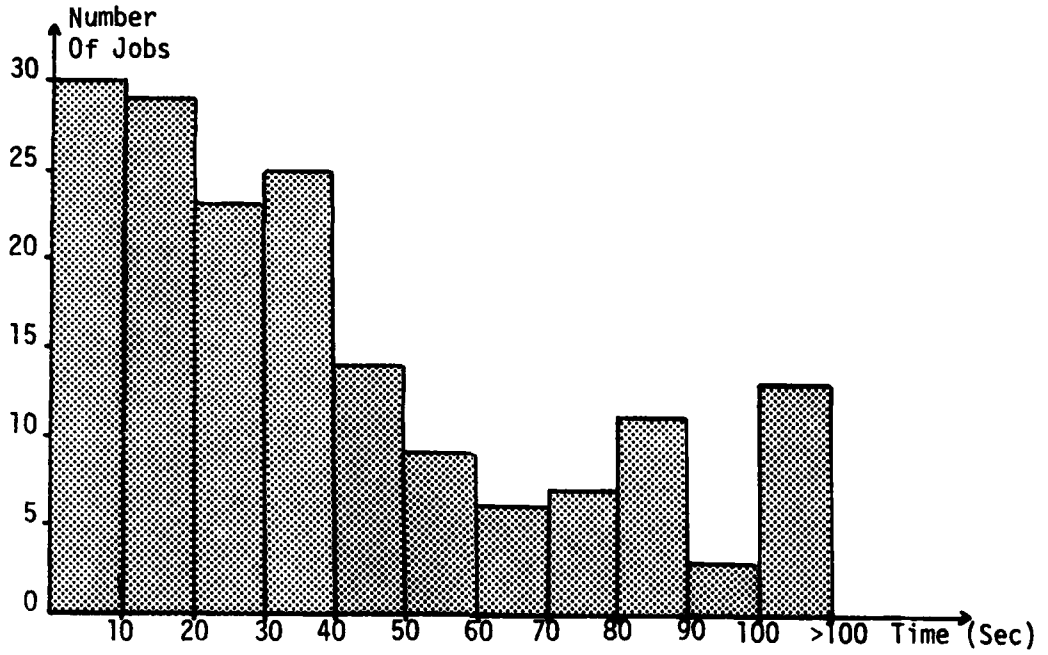
6.4 Analysis of Autobatch Data

There are a total of five Autobatch language translators provided for use with small jobs which require limited I/O support. In addition, a subset of the Statistical Analysis System (SAS76) is provided in-core. The resource demand patterns for jobs executed on these student-oriented translators are very similar. Input data is through the standard input file (card); output is either in printed or punched form; a common region size (256 kilobytes) is used; and access to external files is prohibited. Furthermore, restrictions are placed on the maximum CPU time utilized and maximum output produced.

Due to the similarities in the resource demand characteristics of Autobatch jobs, a limited resource descriptor set is adequate to represent their contribution to the system workload. The descriptor set selected includes CPU time (.01 sec), number of cards read, number of lines printed, and number of cards punched. Data was collected on these four variables during the selected period. Of the 170 Autobatch jobs processed, none punched cards. Thus, this descriptor was eliminated from the set. This resulted in a three variable set denoted hereafter as $\{X_1, X_2, X_3\}$, where $X_1 \equiv$ number of cards read, $X_2 \equiv$ number of lines printed, and $X_3 \equiv$ CPU time in .01 second increments.

The 170 jobs were spread fairly uniformly throughout the period. The interarrival time distribution is depicted in figure 6.5. This distribution is not crucial to the analysis of this section. It must be considered, however, when constructing the final test workload.

Fig. 6.5 Interarrival Distribution - Autobatch



The resource demands of the Autobatch jobs are summarized in table 6.1.

Table 6.1 Resource Demand Characteristics - Autobatch

	X_1	X_2	X_3
Min	4	9	1
Max	873	1328	229
Mean	129.4	162.7	18.4
Std Dev	152.4	203.9	31.8

The variables were first standardized to mean 0, variance 1. Then, the intercorrelations among the variables were examined. These correlations are summarized in table 6.2.

Table 6.2 Correlation Matrix - Autobatch

	X_1	X_2	X_3
X_1	1.000	0.696	0.547
X_2	0.696	1.000	0.758
X_3	0.547	0.758	1.000

The standardized variables X_1,X_2, and X_3 were then subjected to principal component analysis to transform them to the uncorrelated variables Y_1,Y_2, and Y_3. This analysis produced the following linear relations for the composite variables.

$$Y_1 = 0.55052X_1 + 0.60964X_2 + 0.57032X_3$$

$$Y_2 = 0.76719X_1 + 0.10011X_2 + 0.63356X_3$$

$$Y_3 = 0.32915X_1 + 0.78633X_2 + 0.52282X_3$$

The eigenvalues corresponding to the three principal components, the portion of the variability in the data explained by each principal component, and the cumulative portion are displayed in the following table.

Table 6.3 Principal Components for Autobatch Data

	Y_1	Y_2	Y_3
Eigenvalues	2.337483	0.457655	0.204861
Portion	0.779	0.153	0.068
Cum Portion	0.779	0.932	1.000

A table similar to table 6.3 is useful in deciding how many of the

components to retain for the clustering stage. Due to the limited number of variables involved, and the fact that the least significant component (Y_3) accounts for nearly 7% of the variability, no attempt was made to reduce the dimensionality in this case.

It is tempting when using principal component analysis to try to attach a physical meaning to the components. Since in this study, principal components are isolated to examine the bias caused by the intercorrelations and give insight into possible reduction of the dimension of the feature space, no attempt was made to attach such a meaning. It is, however, interesting to note the intercorrelations among the original standardized variables and the component variables. These intercorrelations are shown in table 6.4.

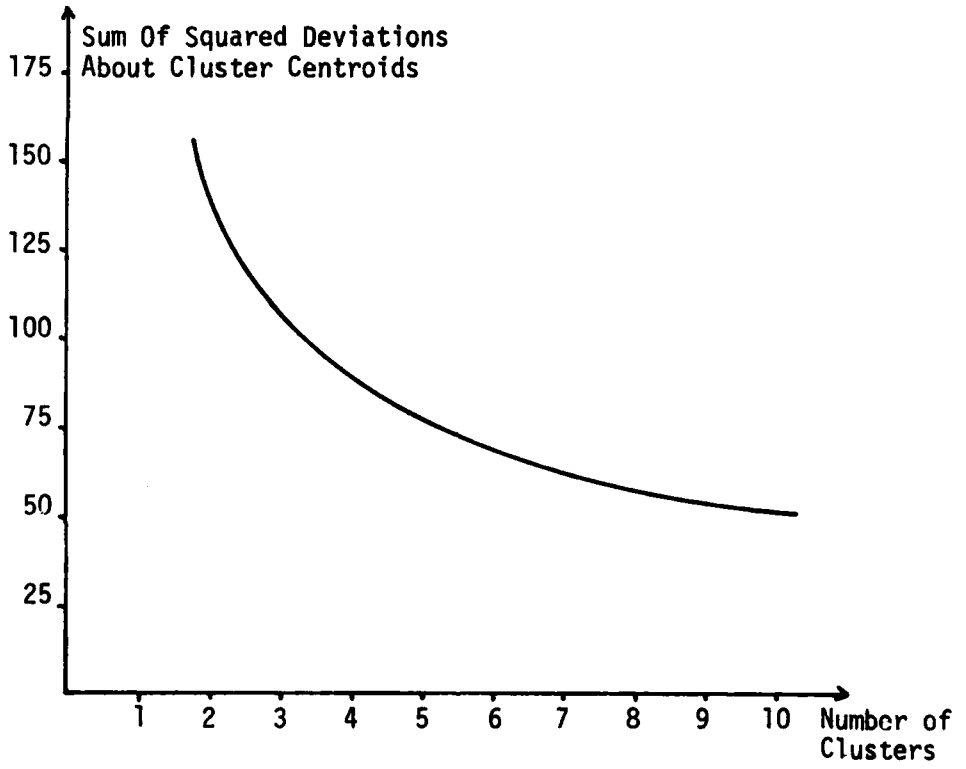
Table 6.4 Intercorrelations Among Variables - Autobatch

	Y_1	Y_2	Y_3
X_1	0.84169	0.51901	0.14898
X_1^2	0.93206	-0.06772	-0.35591
X_3	0.87195	-0.42860	0.23664

Once the component scores were calculated, they were input to the clustering algorithm detailed in Appendix B. The algorithm was run iteratively for various number of clusters, and the sum of the squared deviations about the cluster means examined to determine an appropriate number of clusters. A plot of this measure is depicted in figure 6.6. There is an obvious compromise to be made between obtaining very "tight" clusters and forming the minimum number of clusters necessary. For this

data, a reasonable compromise appeared to be five clusters.

Fig. 6.6 Plot of Cluster "Tightness" - Autobatch



The five clusters formed exhibited markedly different resource demand patterns. To depict the difference, the approximate fractile rankings of the cluster centroids were plotted on Kiviat graphs [59. 69,71]. These graphs, scaled from 0 at the center to 1 on the perimeter, are shown in the following figures.

Fig. 6.7 Kiviat Graph for Cluster 1 - Autobatch

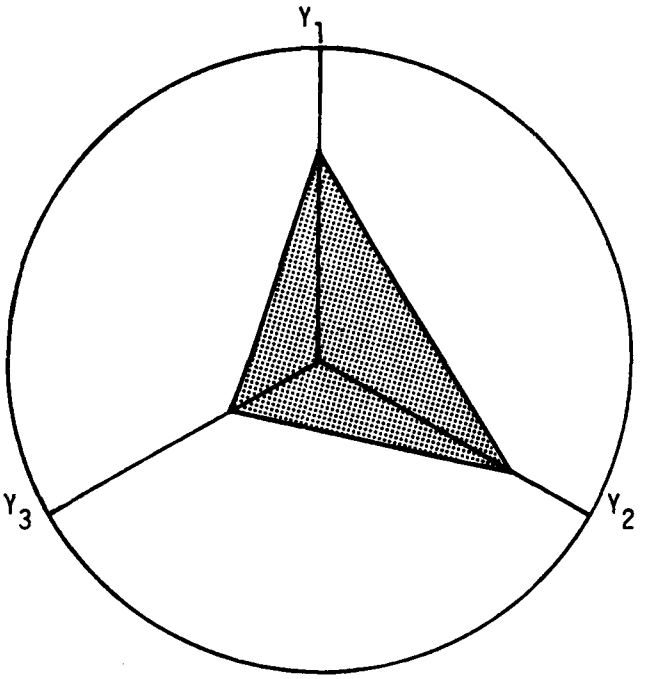


Fig. 6.8 Kiviat Graph for Cluster 2 - Autobatch

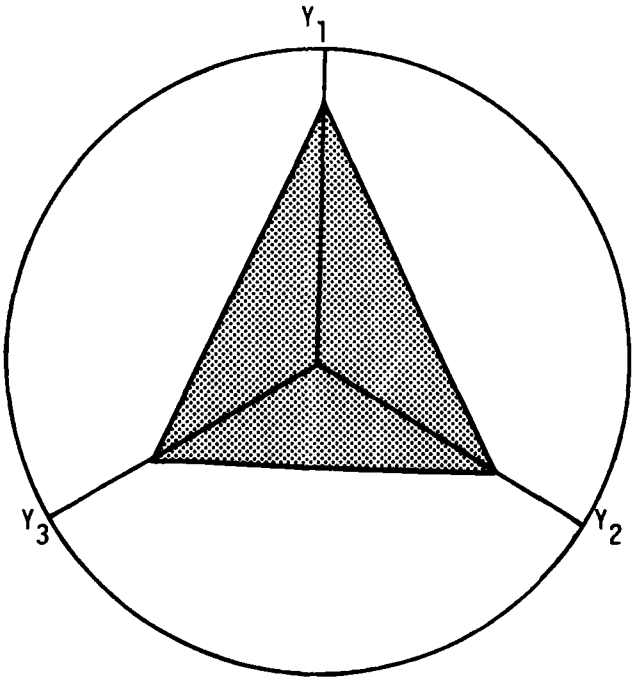


Fig. 6.9 Kiviat Graph for Cluster 3 - Autobatch

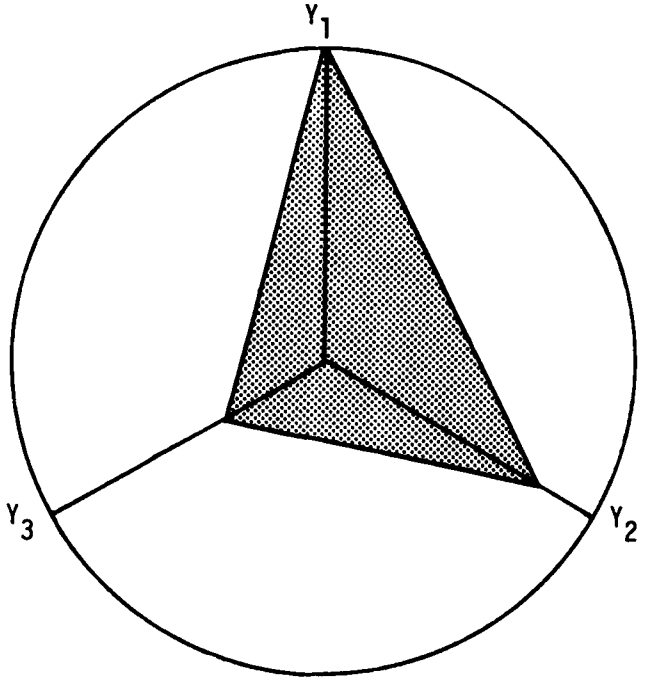


Fig 6.10 Kiviat Graph for Cluster 4 - Autobatch

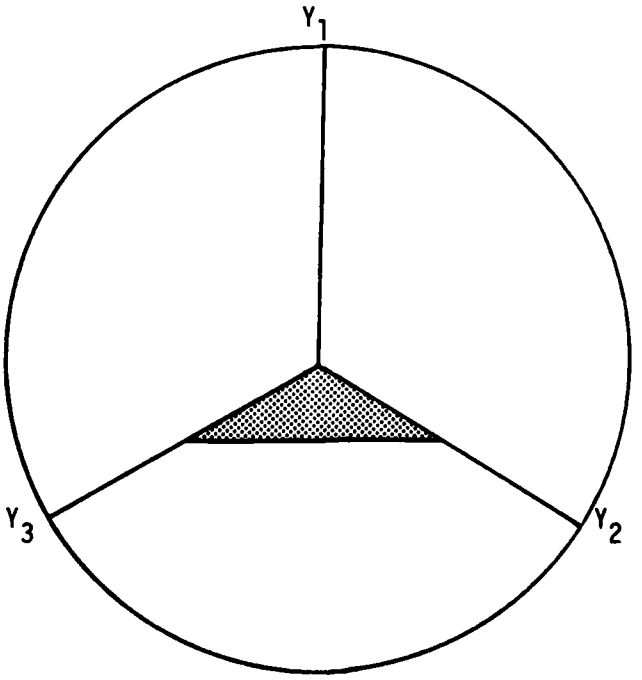
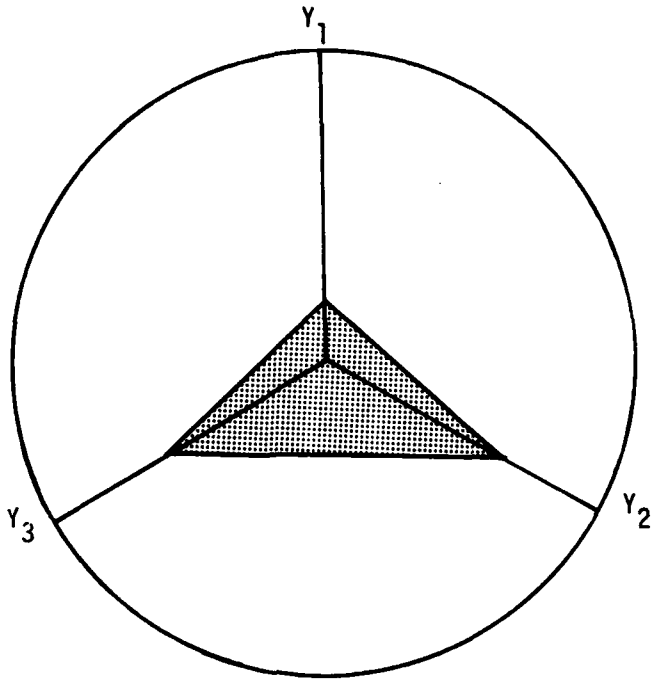


Fig. 6.11 Kiviat Graph for Cluster 5 - Autobatch



Examination of the Kiviat graphs reveals some similarity of structure. For example, both clusters 4 and 5 are severely imbalanced in favor of components Y_2 and Y_3. Clusters 1 and 3, on the other hand, are imbalanced in favor of components Y_1 and Y_2. Similarity of the Kiviat graphs may tempt the analyst to consolidate the two similar clusters into one composite cluster. This may be feasible in some cases, however it should be done with care. The Kiviat graphs display approximate fractile rankings, and, depending upon the variance in the components, a slight difference in the fractile ranking can involve a significant difference in the magnitude of the components.

The interpretation of the clusters in terms of principal components is difficult, since no physical significance was attached to the

components. For this reason, examination of the clusters in the original space is necessary before consolidation of clusters is considered. The cluster characteristics in terms of the original unscaled variables are depicted in table 6.5.

Table 6.5 Cluster Compositions - Autobatch

	Cluster				
	1	2	3	4	5
Number	28	27	16	66	33
X_1(Mean)	130.36	214.85	491.00	29.71	82.64
X_1(std dev)	30.47	109.23	182.60	13.58	22.36
X_2(Mean)	165.64	265.74	664.13	41.74	74.79
X_2(Std dev)	59.67	111.74	253.68	16.04	17.09
X_3(Mean)	13.57	35.22	86.56	3.36	5.91
X_3(Std dev)	9.48	15.23	60.43	1.38	3.54

Examination of table 6.5 reveals that there are indeed significant differences in the magnitude of the demands between the "similar" clusters. No consolidation was attempted for this reason.

6.5 Analysis of Batch Jobs

The restrictions placed upon the allowable resource demands for Autobatch jobs are not applied to jobs using the standard OS translators (Batch jobs). This necessitates an expanded resource descriptor set to adequately characterize Batch jobs, since the range of the resource demands is much broader for these jobs, both in scope and magnitude.

A set of 12 descriptor variables was selected to represent the demands placed on the system by Batch jobs. These are

(a) $X_1 \equiv$ number of job steps executed,

(b) $X_2 \equiv$ total number of devices used by the job,

(c) $X_3 \equiv$ region size requested in kilobytes,

(d) $X_4 \equiv$ number of cards read,

(e) $X_5 \equiv$ number of lines printed,

(f) $X_6 \equiv$ number of cards punched,

(g) $X_7 \equiv$ number of pages read in,

(h) $X_8 \equiv$ number of pages read out,

(i) $X_9 \equiv$ CPU time in .01 second increments,

(j) $X_{10} \equiv$ I/O time in .01 second increments,

(h) $X_{11} \equiv$ EXCP count issued to tape devices, and

(l) $X_{12} \equiv$ EXCP count isued to disk devices (excluding HASP pseudo devices). These 12 variables represent the demands placed upon the major system resources. They also allow discrimination between different types of jobs, such as those which do tape I/O versus disk I/O, or single step versus multistep jobs. An expanded feature set could be used if desired, since reduction of the dimensionality of the feature space is a part of the proposed methodology.

There were a total of 168 Batch jobs processed during the selected period. The interarrival distribution of these jobs is similar to that of the Autobatch jobs as seen in figure 6.12.

The 168 Batch jobs exhibited a widely varying pattern of resource demands as illustrated in table 6.6.

As with the Autobatch data, the variables were first standardized, the correlations examined, and principal component analysis performed. These stages of the analysis are summarized in tables 6.7 and 6.8.

Fig. 6.12 Interarrival Distribution - Batch

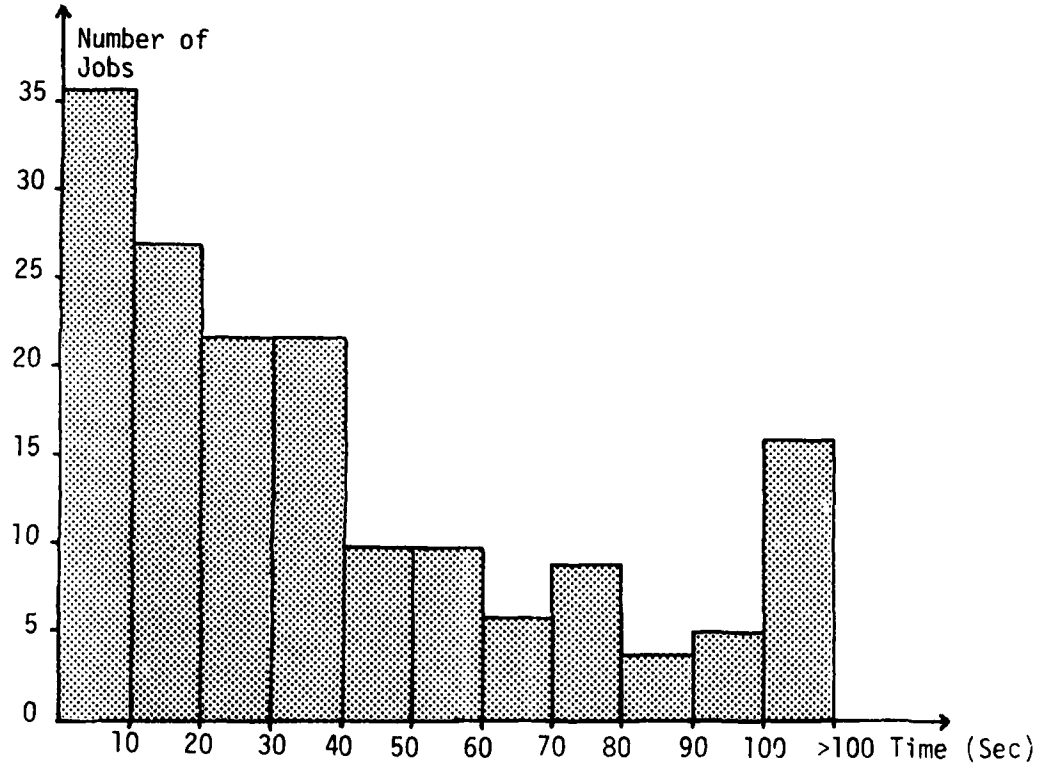


Table 6.6 Resource Demand Characteristics - Batch

	Min	Max	Mean	Std Dev
X_1	1	6	1.56	1.03
X_2	2	61	12.18	10.68
X_3	64	512	159.24	80.76
X_4	5	4619	257.87	655.33
X_5	0	24979	1872.69	4771.51
X_6	0	6548	76.11	674.57
X_7	0	440	31.73	49.16
X_8	0	384	12.43	34.58
X_9	2	12731	336.39	1193.59
X_{10}	0	29998	812.40	2948.55
X_{11}	0	33028	354.67	2612.71
X_{12}	0	47677	1009.78	4556.96

Table 6.7 Correlation Matrix - Batch

	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}
X_1	1.00	0.59	0.37	0.08	-0.05	-0.06	0.33	0.29	0.08	0.08	0.07	0.12
X_2	0.59	1.00	0.42	0.13	0.08	-0.10	0.59	0.39	0.28	0.24	0.20	0.34
X_3	0.37	0.42	1.00	0.02	-0.02	-0.13	0.31	0.21	0.16	0.07	0.04	0.19
X_4	0.08	0.13	0.02	1.00	0.01	-0.04	0.00	-0.01	0.00	-0.05	-0.04	-0.03
X_5	-0.05	0.08	-0.02	0.01	1.00	-0.04	0.21	0.25	0.17	-0.01	-0.04	0.25
X_6	-0.06	-0.10	-0.13	-0.04	-0.04	1.00	-0.05	-0.04	-0.03	-0.03	-0.02	-0.02
X_7	0.33	0.59	0.31	0.00	0.21	-0.05	1.00	0.89	0.52	0.43	0.32	0.76
X_8	0.29	0.39	0.21	-0.01	0.25	-0.04	0.89	1.00	0.50	0.34	0.25	0.84
X_9	0.08	0.28	0.16	0.00	0.17	-0.03	0.52	0.50	1.00	0.82	0.85	0.41
X_{10}	0.08	0.24	0.07	-0.05	-0.01	-0.03	0.43	0.34	0.82	1.00	0.85	0.23
X_{11}	0.07	0.20	0.04	-0.04	-0.04	-0.02	0.32	0.25	0.85	0.85	1.00	0.08
X_{12}	0.12	0.34	0.19	-0.03	0.25	-0.02	0.76	0.84	0.41	0.23	0.08	1.00

Table 6.8 Principal Components for Batch Data

	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	Y_{10}	Y_{11}	Y_{12}
Eigenvalues	4.22	1.97	1.47	1.03	0.98	0.80	0.66	0.39	0.20	0.16	0.07	0.06
Portion	0.35	0.16	0.12	0.09	0.08	0.07	0.06	0.03	0.02	0.01	0.01	0.00
Cam Portion	0.35	0.51	0.63	0.72	0.80	0.87	0.93	0.96	0.98	0.99	1.00	1.00

Examination of table 6.8 shows that 96% of the total variance in the data can be explained by retaining only 8 of the 12 components. These 8 most significant components were selected to be input to the clustering algorithm. The intercorrelations among these 8 most significant components and the 12 original variables is shown in table 6.9.

Table 6.9 Intercorrelations Among Variables - Batch

	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8
X_1	0.40	0.47	0.54	-0.16	0.07	0.18	-0.37	0.36
X_2	0.63	0.41	0.39	-0.01	0.09	0.17	-0.18	-0.46
X_3	0.38	0.40	0.43	-0.08	-0.23	0.13	0.65	0.07
X_4	0.01	0.17	0.22	0.61	0.70	-0.22	0.13	0.05
X_5	0.22	0.15	-0.53	0.41	0.00	0.69	0.00	0.05
X_6	-0.09	-0.13	-0.19	-0.66	0.65	0.24	0.15	-0.01
X_7	0.89	0.25	-0.18	-0.07	0.02	-0.13	-0.04	-0.09
X_8	0.83	0.25	-0.36	-0.06	0.02	-0.20	-0.05	0.16
X_9	0.80	-0.51	0.06	0.09	0.02	0.07	0.11	0.06
X_{10}	0.69	-0.63	0.17	0.02	-0.01	0.01	-0.02	-0.01
X_{11}	0.61	-0.70	0.25	0.04	0.01	0.07	-0.03	0.02
X_{12}	0.71	0.29	-0.49	-0.07	-0.01	-0.26	0.07	0.00

To determine a reasonable number of clusters to form, a procedure similar to that used with the Autobatch data was followed. The plot of the total summed deviations about the cluster means is shown in figure 6.13.

Based upon the plot of figure 6.13, a reasonable compromise appeared to be to form 10 clusters. The approximate fractile rankings of the cluster centroids are depicted in the following Kiviat graphs.

Fig. 6.13 Plot of Cluster "Tightness" - Batch

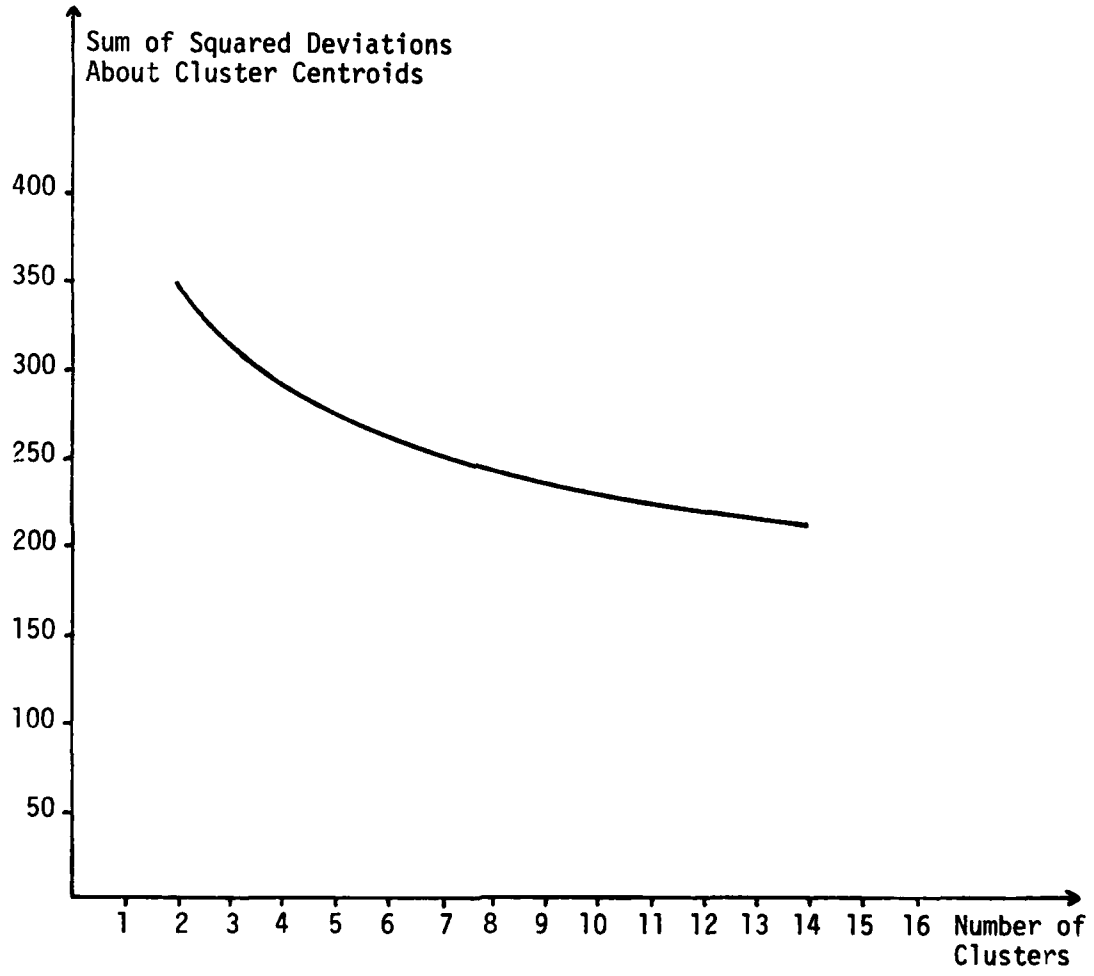


Fig. 6.14 Kiviat Graph for Cluster 1 - Batch

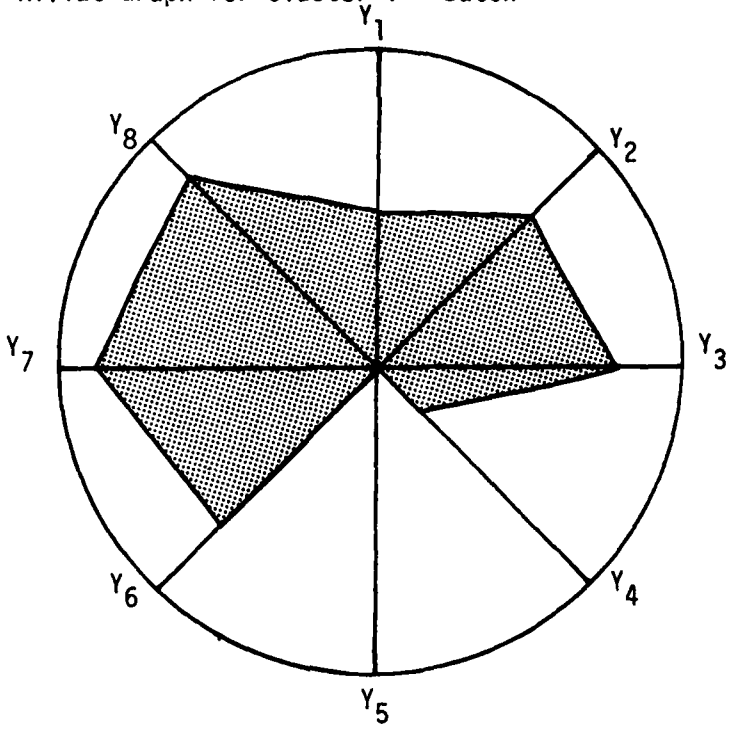


Fig. 6.15 Kiviat Graph for Cluster 2 - Batch

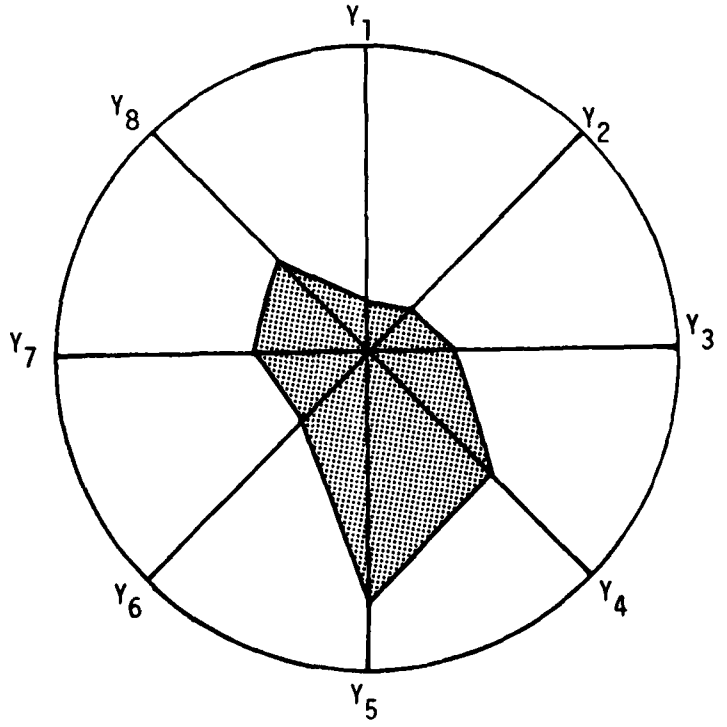


Fig. 6.16 Kiviat Graph for Cluster 3 - Batch

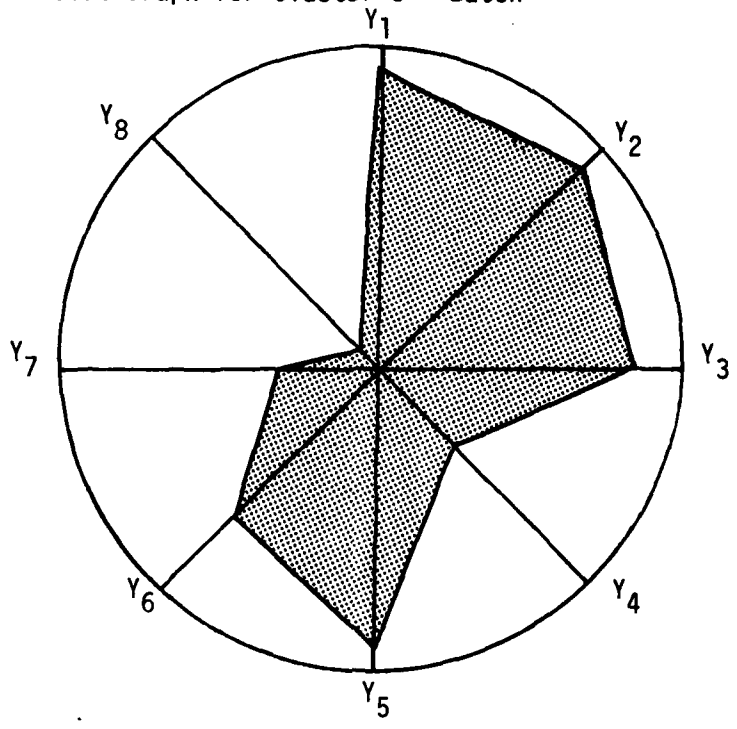


Fig. 6.17 Kiviat Graph for Cluster 4 - Batch

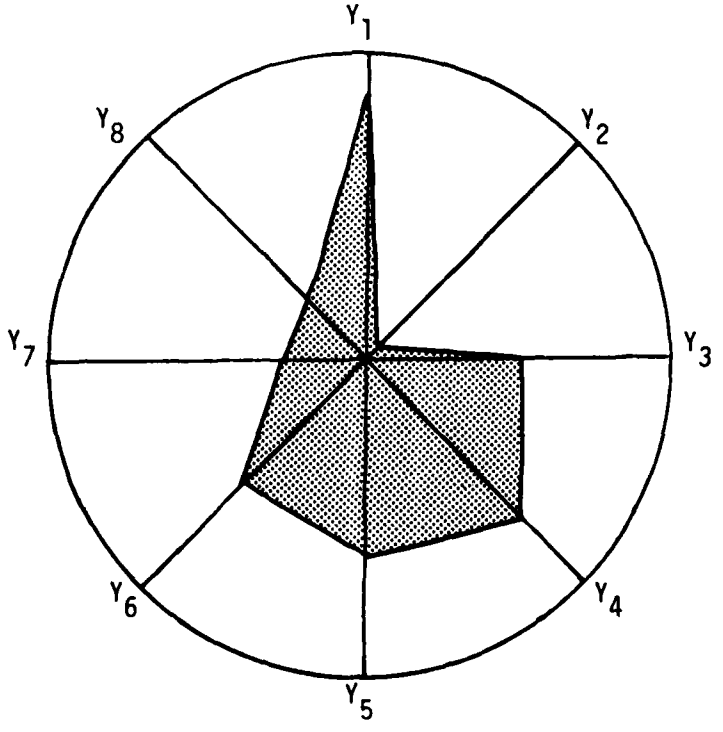


Fig. 6.18 Kiviat Graph for Cluster 5 - Batch

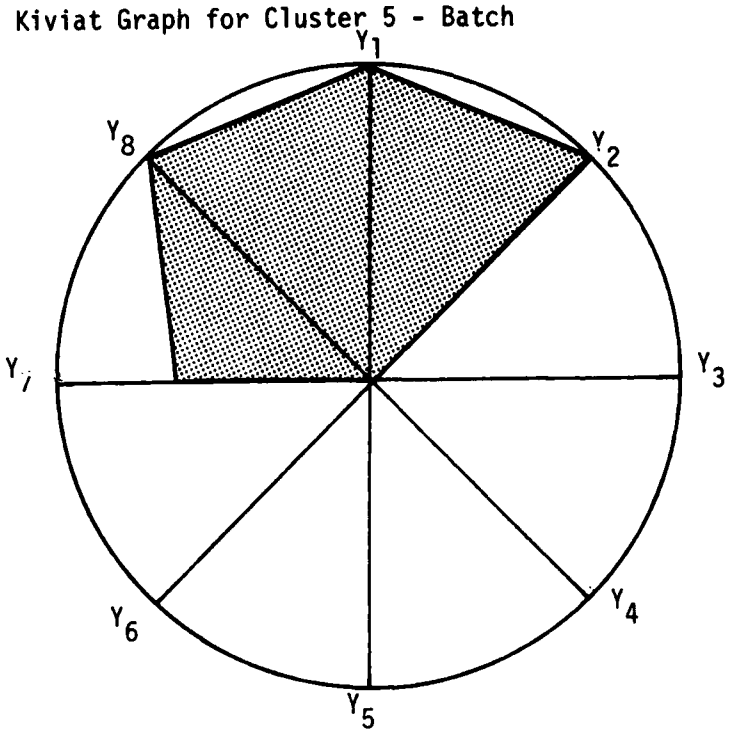


Fig. 6.19 Kiviat Graph for Cluster 6 - Batch

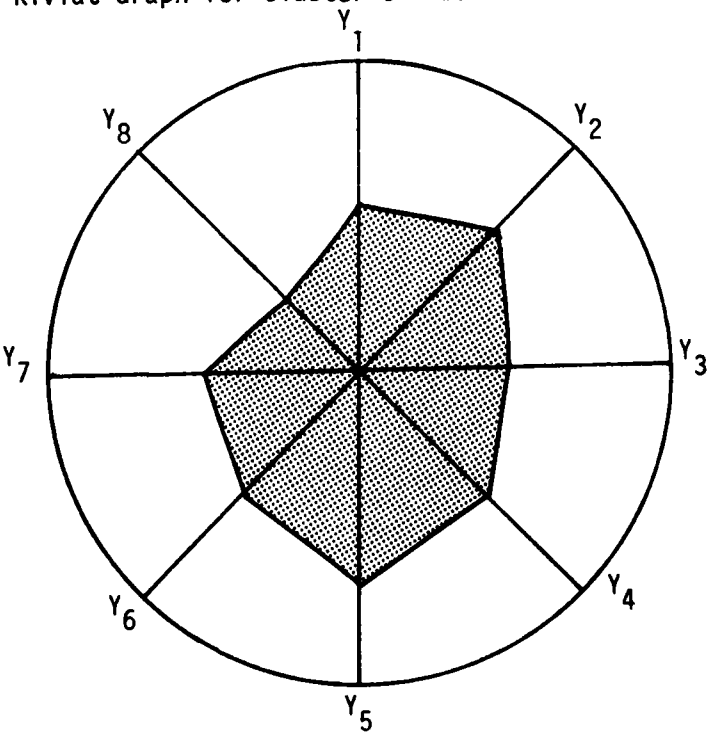


Fig. 6.20 Kiviat Graph for Cluster 7 - Batch

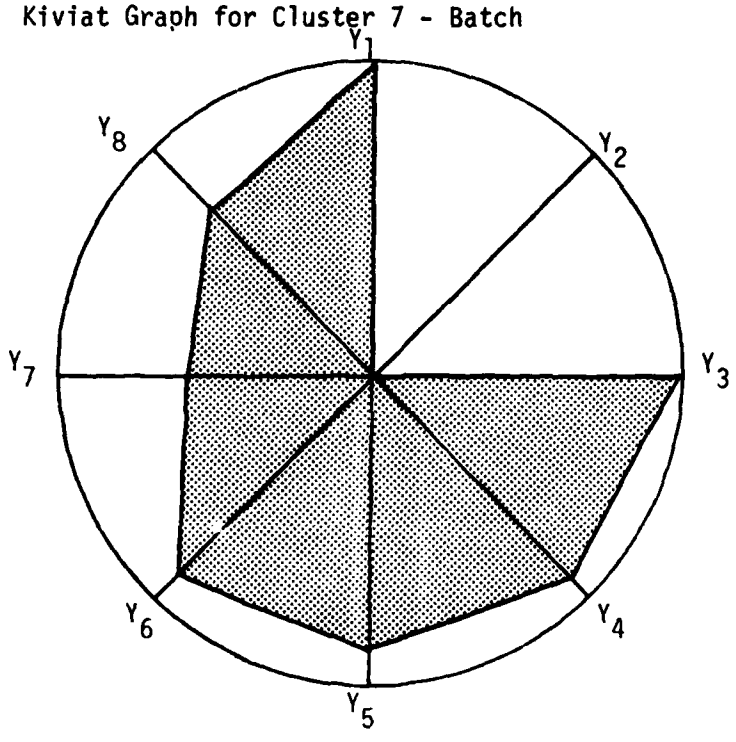


Fig. 6.21 Kiviat Graph for Cluster 8 - Batch

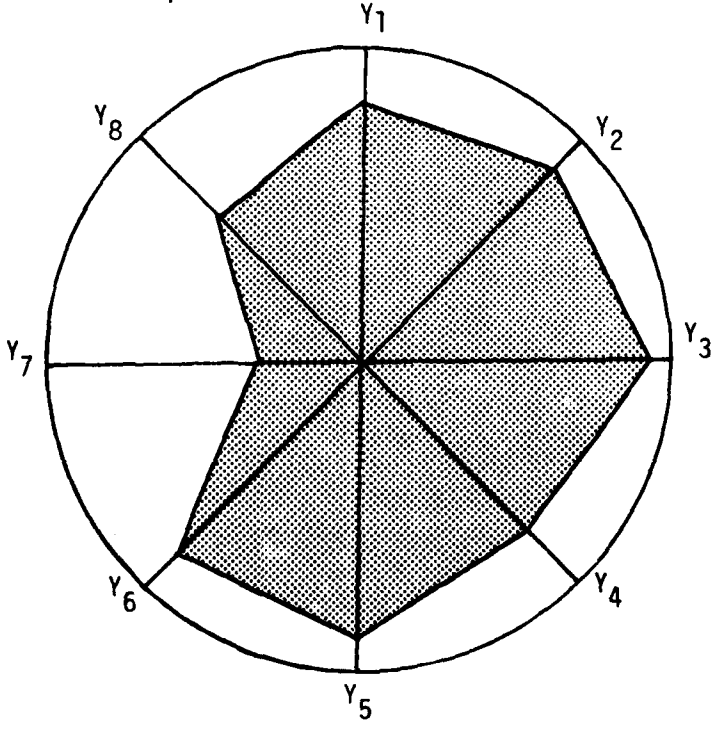


Fig. 6.22 Kiviat Graph for Cluster 9 - Batch

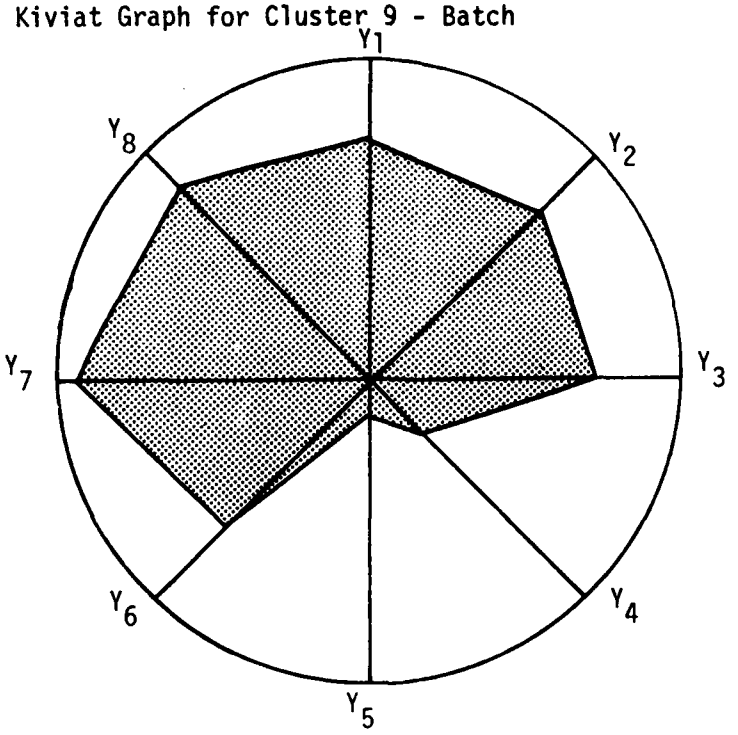
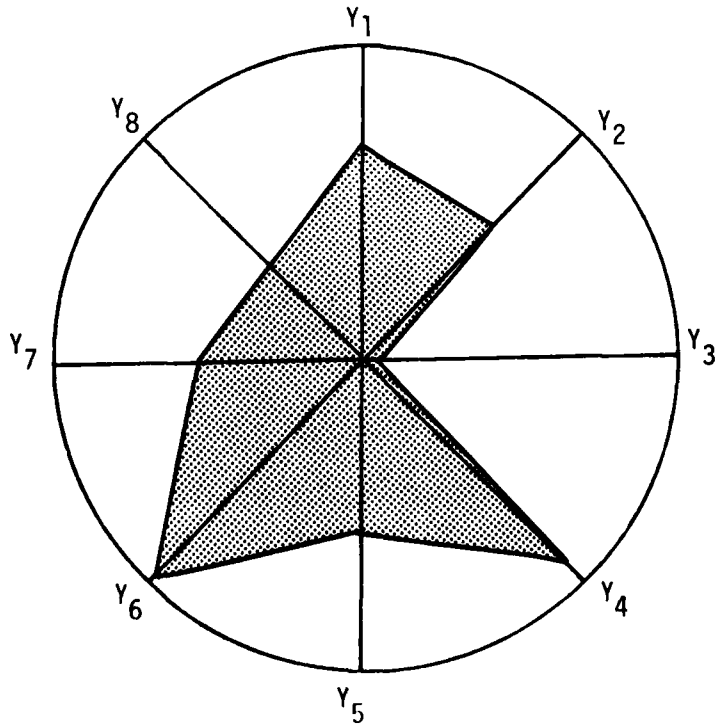


Fig. 6.23 Kiviat Graph for Cluster 10 - Batch



The Kiviat graphs show a distinct structure for each cluster, thus it is unlikely that consolidation of any of the clusters would be beneficial. The cluster compositions in terms of the original variables are shown in table 6.10.

6.6 Comparison of Clustering Results

The intercorrelation among the resource descriptor variables biases the results of the clustering phase of analysis. Various weighting schemes were proposed in chapter IV to neutralize this bias. In this section, the clusters achieved when these weighting schemes were applied to the Batch workload data will be compared. Similar experiments were conducted using the Autobatch data with comparable results. Toward the end of the section, the clustering results achieved by retaining the eight most significant components will be compared to those which were achieved by retaining all 12 of the components, using the same weighting scheme in both cases.

The Batch workload data was standardized and then subjected to principal components analysis. The component scores (all components retained) were then input to the clustering algorithm detailed in Appendix B with three different weighting schemes. The first run used an unweighted Euclidean distance metric. The second two runs used a weighted Euclidean distance metric with $W_i = 1/\lambda_i$ in one case and $W_i = \lambda_i$ in the other case.

It is difficult to compare the partitions achieved using different weighting schemes since the cluster memberships can change quite drastically. Since the aim of applying the weighting function was to

Table 6.10 Cluster Compositions - Batch

	Cluster									
	1	2	3	4	5	6	7	8	9	10
Number	12	55	10	4	1	46	1	20	12	7
X_1^1(Mean)	1.83	1.00	3.10	1.25	3.00	1.28	2.00	3.10	1.33	1.00
X_1^1(Std dev)	0.37	0.00	1.97	0.43	0.00	0.50	0.00	1.09	0.47	0.00
X_2^1(Mean)	7.08	4.18	40.20	12.50	23.00	10.39	32.00	22.55	18.25	10.86
X_2^2(Std dev)	0.28	2.03	11.32	7.12	0.00	4.55	0.00	6.20	6.04	5.79
X_3^2(Mean)	272.00	97.75	249.60	112.00	256.00	139.13	192.00	201.60	282.67	128.00
X_3^2(Std dev)	27.71	42.02	66.82	27.71	0.00	30.73	0.00	73.74	80.36	0.00
X_4^2(Mean)	46.25	244.00	522.10	53.25	18.00	238.09	17.00	566.40	87.75	77.86
X_4^4(Std dev)	37.71	713.43	837.72	56.83	0.00	539.67	0.00	977.69	74.52	100.70
X_5^4(Mean)	240.08	389.93	2497.30	2252.50	17713.00	1047.28	192.00	1365.05	638.42	22180.00
X_5^5(Std dev)	281.90	1183.67	2376.45	3491.06	0.00	1663.43	0.00	1678.17	822.36	3263.14
X_6^5(Mean)	0.00	232.47	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
X_6^6(Std dev)	0.00	1159.88	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
X_7^6(Mean)	8.67	8.85	129.50	66.25	440.00	23.09	199.00	37.20	50.67	18.14
X_7^7(Std dev)	6.24	9.39	35.54	28.89	0.00	15.32	0.00	32.52	26.01	18.50
X_8^7(Mean)	0.00	1.69	53.80	19.50	384.00	7.85	110.00	16.30	12.50	7.00
X_8^8(Std dev)	0.00	3.64	28.11	16.10	0.00	9.51	0.00	21.62	9.48	11.50
X_9^8(Mean)	31.75	26.11	1485.50	447.25	4307.00	176.09	12731.00	304.95	139.17	735.14
X_9^9(Std dev)	15.18	51.94	1975.06	331.69	0.00	282.49	0.00	320.39	169.96	242.84
X_{10}^9(Mean)	58.75	161.65	3296.80	7402.75	3592.00	311.93	29998.00	522.40	388.58	179.71
X_{10}^{10}(Std dev)	19.04	812.63	3375.37	5950.66	0.00	481.56	0.00	347.03	537.70	169.45
X_{11}^{10}(Mean)	0.00	18.65	1087.90	2578.25	0.00	77.80	33028.00	38.00	0.00	0.00
X_{11}^{11}(Std dev)	0.00	111.71	1379.92	1422.34	0.00	274.20	0.00	110.22	0.00	0.00
X_{12}^{11}(Mean)	20.33	74.05	8108.50	1045.50	47677.00	240.96	3975.00	551.80	352.00	294.71
X_{12}^{12}(Std dev)	43.02	228.13	8058.72	1510.70	0.00	294.09	0.00	653.74	434.11	389.47

more or less equalize the intracluster variation in each dimension, one way to compare the results obtained is to examine the variance (or standard deviation) for each variable within each cluster. It is likely not possible nor desirable to achieve true equality. This is because of the wide disparity in the variances of the principal component variables when all data units are considered (i.e. Var (Y_1) = 4.22; Var (Y_{12}) = 0.06). It should be apparent, however, that more homogeneous clusters are formed if the intracluster variations are small and nearly equal.

Since the clustering algorithm was applied to the principal component scores, any comparison made is most meaningful in the principal components space. The intracluster standard deviations for each principal component variable within each cluster are shown in tables 6.11, 6.12, and 6.13.

Table 6.11 Cluster Standard Deviations - $W_i = 1$

	1	2	3	4	5	6	7	8	9	10
Y_1	.46	1.05	8.74	2.95	3.67	1.01	0.00	2.41	.55	.84
Y_2	.20	.75	2.43	2.09	1.74	.45	0.00	1.02	.24	.49
Y_3	.24	.52	3.88	1.24	1.03	.52	0.00	1.05	.13	.40
Y_4	.04	1.01	.32	.37	.94	.17	0.00	.48	.07	.28
Y_5	.10	.97	.10	.59	.69	.28	0.00	.39	.10	.17
Y_6	.10	.42	.95	.19	.73	.22	0.00	.42	.09	.47
Y_7	.35	.30	.16	.66	.79	.69	0.00	.79	.14	.12
Y_8	.14	.22	1.14	.28	.28	.35	0.00	.54	.16	.26
Y_9	.06	.10	.39	.39	.10	.17	0.00	.32	.07	.11
Y_{10}	.03	.20	.15	.15	.06	.07	0.00	.06	.02	.04
Y_{11}	.02	.03	.15	.24	.08	.03	0.00	.05	.01	.04
Y_{12}	.03	.04	.09	.11	.03	.09	0.00	.06	.03	.03

Table 6.12 Cluster Standard Deviations - $W_i = 1/\lambda_i$

	1	2	3	4	5	6	7	8	9	10
Y_1	0.00	3.42	3.08	2.03	1.73	1.27	13.60	5.36	.97	.84
Y_2	0.00	1.08	1.32	.85	.87	.51	6.15	2.11	.73	.49
Y_3	0.00	1.18	2.21	1.18	.93	.34	2.05	1.27	.78	.40
Y_4	0.00	.93	.72	.38	.84	.16	.35	.47	.24	.28
Y_5	0.00	.90	1.01	.39	.93	.13	.23	.30	.31	.17
Y_6	0.00	.48	.62	.21	.50	.25	.62	.51	.29	.47
Y_7	0.00	.34	.74	1.01	.57	.43	.35	1.02	.28	.12
Y_8	0.00	.28	.36	.37	.08	.29	.08	.42	.18	.26
Y_9	0.00	.11	.41	.17	.07	.13	.10	.33	.08	.11
Y_{10}	0.00	.07	.05	.13	.05	.02	1.11	.09	.03	.04
Y_{11}	0.00	.03	.14	.04	.07	.03	.07	.22	.02	.04
Y_{12}	0.00	.04	.08	.09	.02	.05	.06	.04	.02	.03

Table 6.13 Cluster Standard Deviations - $W_i = \lambda_i$

	1	2	3	4	5	6	7	8	9	10
Y_1	.21	.42	2.74	1.01	0.00	.84	0.00	.50	.59	.85
Y_2	.20	.46	2.64	1.02	0.00	.89	0.00	.49	.39	.49
Y_3	.22	.52	2.24	.90	0.00	.41	0.00	.69	.46	.40
Y_4	.07	1.33	.85	.36	0.00	.52	0.00	1.27	.14	.28
Y_5	.06	1.31	1.03	.24	0.00	.58	0.00	1.12	.35	.17
Y_6	.04	.47	.69	.31	0.00	.35	0.00	.45	.14	.47
Y_7	.29	.40	.96	.71	0.00	.40	0.00	.99	.93	.12
Y_8	.13	.12	.81	.42	0.00	.29	0.00	.45	.46	.26
Y_9	.04	.05	.52	.13	0.00	.13	0.00	.15	.11	.11
Y_{10}	.01	.08	.10	.15	0.00	.26	0.00	.06	.07	.04
Y_{11}	.01	.02	.26	.10	0.00	.04	0.00	.05	.02	.04
Y_{12}	.02	.03	.09	.12	0.00	.05	0.00	.05	.06	.03

Close examination of tables 6.11, 6.12, and 6.13 tends to

confirm the conclusions of Chapter IV, particularly if viewed in terms of the extreme values of the cluster standard deviations. Table 6.11, based upon the unweighted distance metric, has a maximum value of 8.74, and a total of 16 values greater than 1. Table 6.12, based upon the weighted distance metric with $W_i = 1/\lambda_i$, has a maximum value of 13.60, and a total of 20 values greater than 1. Table 6.13, based upon the weighted distance metric with $W_i = \lambda_i$, has a maximum value of 2.74, and a total of 9 values greater than 1. Thus, the weighting scheme with $W_i = 1/\lambda_i$ actually performs worse than the unweighted scheme, while significant improvement is noted when $W_i = \lambda_i$ is used.

Of particular note with this data is the manner in which the three schemes handled outlier jobs. There were two jobs which were much larger in terms of resource requirements than any others in the subset. Both jobs performed an excessive amount of I/O, with one accessing tape devices and the other disk devices. Both the unweighted version and the weighted version with $W_i = 1/\lambda_i$ grouped at least one of these outlier jobs with other data units, thus providing a very inhomogeneous cluster. Only the weighted scheme with $W_i = \lambda_i$ "correctly"classified these two jobs into two single member clusters.

Comparison of the results obtained with a weighted distance metric ($W_i = \lambda_i$) when 8 and 12 of the principal components are retained indicate very little change. Of the 10 clusters obtained with 12 components, 5 of them remain intact when only 8 components are retained (including the two "outlier" clusters mentioned above). There are but minor changes in 4 of the 5 remaining clusters. The lone cluster which changed drastically was a small cluster (9 data units) in which

even minor alterations in cluster membership can have a dramatic effect on cluster characteristics. In all, 15 of the 168 data units migrated (i.e. changed clusters) when the four least significant components were dropped. This performance is somewhat related to the weighting scheme used. Similar experiments were conducted using the unweighted distance metric and the weighted distance metric ($W_i = 1/\lambda_i$). The effect of dropping the four least significant components was more severe with these two schemes.

6.7 Construction of Synthetic Jobs

The construction of synthetic jobs to replace the real jobs in the selected workload subset is the next logical step following clustering. A separate synthetic job is generally required to represent each cluster. There may be exceptions to this however. Two clusters may be similar enough that a single synthetic job can be used to represent the jobs in the composite cluster formed by merging the two. A single cluster, on the other hand, may be too "loose" to allow adequate representation of its members with a single synthetic job. Such a cluster must be split into subclusters, each of which is represented by a separate synthetic job. After synthetic jobs are constructed for each cluster/subcluster, the synthetic mix can be formed by including the appropriate number of copies of each job and appending the arrival time to each.

Synthetic jobs were constructed for one Autobatch cluster and one Batch cluster to illustrate the design technique. The Autobatch cluster selected was cluster 4 (see table 6.5 (p.91)), while the Batch cluster selected was cluster 6 (see table 6.10 (p.103)).

Synthetic jobs designed to represent the Autobatch jobs can be very simple jobs due to the limited resource descriptor set. Three resource demands must be controlled: $X_1 \equiv$ number of cards read, $X_2 \equiv$ number of lines printed, and $X_3 \equiv$ CPU time used (.01 sec). The number of cards read is exactly determined by the number of source/comment statements in the program and the number of JCL/data cards. This number can be varied within certain limits for a given synthetic job by either including or excluding data/comment cards. The number of lines printed can also be exactly controlled by including a print loop which is executed the desired number of times. CPU time used is controlled by executing a compute loop a certain number of times. The amount of CPU time is also related to the number of lines printed, hence this dependence must be accounted for. The synthetic job designed for Autobatch cluster 4 is described in detail in Appendix D.

The synthetic job for Autobatch cluster 4 has two parameters which may be varied to induce various resource demand patterns. These parameters are NRLIN ≡ the number of lines to be printed and NITER ≡ the number of times the compute loop is to be executed. The size of the program (number of cards read) was held constant throughout. These two parameters were used as "treatments" in the experimental design used. Three "levels" for each "treatment" were established to cover the range of resource demands exhibited by the members of Autobatch cluster 4. This results in nine unique treatment/level combinations. A completely randomized factorial design (3^2) was used to establish the parameter settings for the nine required runs of the job. The parameter setting for each run are shown in table 6.14.

Table 6.14 Parameter Settings - Autobatch

	Run								
	1	2	3	4	5	6	7	8	9
NRLIN	50	50	50	150	150	0	0	150	0
NITER	50	5000	2500	50	2500	5000	50	5000	2500

The nine programs were run on the system and data collected which reflected the resource demands of each program. This data is summarized in table 6.15.

Table 6.15 Resource Demands - Synthetic Autobatch Job

	Run								
	1	2	3	4	5	6	7	8	9
X_1	33	33	33	33	33	33	33	33	33
X_2	88	88	88	188	188	38	39	188	38
X_3	5	77	41	9	43	74	3	80	38

The significance of the effect of varying NITER and NRLIN on X_3 was then tested. Both "treatments" were found to be highly significant (α = .0001). The model used assumed no interaction between the parameters. The amount of CPU time used (X_3) was regressed on NITER and NRLIN, while the number of lines printed (X_2) was regressed on NRLIN. The following predictor equations were obtained through this regression:

$$X_2 = 38.238 + 0.998 \text{ NRLIN}$$

$$X_3 = 2.399 + 0.037 \text{ NRLIN} + 0.014 \text{ NITER}.$$

The fit achieved by both regression equations was extremely good. The value of the multiple correlation coefficient (proportion of the variability explained) was 0.999978 for the equation relating X_2 to NRLIN, and 0.999679 for the equation relating X_3 to NRLIN and NITER.

Synthetic jobs designed to represent Batch jobs must be considerably more complex than those for Autobatch jobs due to the expanded resource descriptor set. The descriptor set used for the Batch jobs includes 12 variables. A number of these can be exactly controlled through Job Control Language (JCL) statements or the inclusion /exclusion of data/comment cards. Others must be controlled through parameters.

The synthetic job designed for Batch cluster 6 (described in Appendix D) has four parameters which can be varied to induce different resource demand patterns. They are NITER ≡ the number of times the compute loop is executed, NOUT ≡ the number of output lines produced, NTAP ≡ the number of records read from a tape file, and NDIS ≡ the number of records read from a disk file. Those resource demands which are not affected by varying these parameters were held constant throughout the experiment.

Two levels for each parameter were selected. A completely randomized factorial design (2^4) was used to establish the parameter settings for the various runs of the program. This design requires 16 runs to form one replication of the experiment. This was considered excessive due to the cost associated with each run. It was decided to use a fractional replication for this reason. A one half fractional replication requires only eight runs, but still allows testing of the

main treatment effects. The effect of interaction among parameters was assumed negligible just as with the Autobatch experiment. Using the method illustrated in Hicks [43], the "treatment" combinations were divided into two blocks, with the four-way interaction effect confounded with the block effect. A coin flip was used to decide which of the blocks to use in the experiment. The parameter settings for the eight required runs of the job are listed in table 6.16.

Table 6.16 Parameter Settings - Batch

	Run							
	1	2	3	4	5	6	7	8
NITER	1000	0	1000	0	0	0	1000	1000
NOUT	0	0	1000	0	1000	1000	0	1000
NTAP	0	1000	1000	0	0	1000	1000	0
NDIS	0	0	0	1000	0	1000	1000	1000

The synthetic jobs were run on the system, and data collected reflecting the resource demands. This data is shown in table 6.17. The values for all 12 resource descriptors are shown; those which are not affected by the four parameters appear as constants. No attempt was made to control paging behavior as this is largely environment dependent.

Table 6.17 shows that five of the 12 resource descriptors are affected by varying the four parameters. These are X_5 (number of lines printed), X_9 (CPU time used in .01 sec increments), X_{10} (I/O time used in .01 sec increments), X_{11} (EXCP count to tape devices), and X_{12} (EXCP count to disk devices). The significance of the effect of the parameters

Table 6.17 Resource Demands - Synthetic Batch Job

	Run 1	2	3	4	5	6	7	8
X_1	1	1	1	1	1	1	1	1
X_2	14	14	14	14	14	14	14	14
X_3	128	128	128	128	128	128	128	128
X_4	240	240	240	240	240	240	240	240
X_5	354	351	1349	351	1349	1349	351	1349
X_6	0	0	0	0	0	0	0	0
X_7	0	0	0	0	0	0	0	0
X_8	0	0	0	0	0	0	0	0
X_9	242	109	332	111	191	201	247	329
X_{10}	188	213	212	233	187	256	257	232
X_{11}	1	10	10	1	1	10	10	1
X_{12}	132	132	132	217	132	217	217	217

on the descriptor variables was tested. Using a level of significance $\alpha = .05$, the effect on X_9 was significant for NITER and NOUT; the effect on X_{10} was significant for NOUT, NTAP, and NDIS; the effect on X_5 was significant for NOUT; the effect on X_{11} was significant for NTAP; and the effect on X_{12} was significant for NDIS.

The descriptor variables were then regressed on those parameters which were identified as having a statistically significant effect. The resulting regression equations with the value of the multiple correlation coefficient indicated in parentheses are

$X_5 = 351.75 + 0.99725\text{NOUT}$ ($R^2 = 0.999997$),

$X_9 = 110.00 + 0.1345\text{NITER} + 0.0860\text{NOUT}$ ($R^2 = 0.998648$),

$X_{10} = 188.25 - 0.001\text{NOUT} + 0.025\text{NTAP} + 0.0445\text{NDIS}$ ($R^2 = 0.999903$),

$X_{11} = 1.00 + 0.009\text{NTAP}$ ($R^2 = 1.000000$), and

$X_{12} = 132.00 + 0.085\text{NDIS}$ ($R^2 = 1.000000$).

The problem with inverting the above equations to yield predictor

equations for the parameter settings is that there is one equation too many (i.e. 5 equations in 4 unknowns). The equation for X_{10} however is seen to be redundant, since I/O time is uniquely determined by the quantity and type of I/O performed. Inverting the remaining relations yields the following predictor equations

$$NOUT = 1.00276X_5 - 352.72,$$

$$NITER = 7.4349X_9 - 0.6411X_5 - 592.27,$$

$$NTAP = 111.1111X_{11} - 111.11, \text{ and}$$

$$NDIS = 11.7647X_{12} - 1552.95.$$

6.8 Summary

A statistical methodology proposed for use in constructing test workloads was developed in Chapters III, IV, and V. The major elements of this methodology are illustrated in this chapter with a detailed case study of the workload processed by the Amdahl 470/V6 at Texas A&M University.

The first task in constructing a test workload is determining a subset of the real workload to use as a model. The appropriate workload subset is related to the particular evaluation study being performed. An overall workload profile can be constructed, and an applicable subset selected by viewing the characteristics displayed in the profile. This study was not directed toward any particular evaluation effort, hence the choice of the subset was somewhat arbitrary.

The selected workload subset was found to be composed of two basic types of jobs, those using the student compilers (Autobatch) and those using the standard OS translators (Batch). A limited

resource descriptor set is adequate for characterizing the resource demands of the Autobatch jobs while an expanded set is required for Batch jobs. The two types of jobs were analyzed separately for this reason.

The resource demand matrix was first scaled so that each descriptor variable had a mean of 0 and a variance of 1. This scaled matrix was then subjected to principal component analysis, to transform the demand vectors to a space of uncorrelated composite variables. Those component variables necessary to explain 95% of the total variability were retained. This resulted in the retention of all three of the component variables for the Autobatch data. Only eight of the 12 component variables for the Batch data were retained, however, reducing the dimensionality of the problem by one third.

The principal component scores were input to a non-hierarchical clustering algorithm using a weighted Euclidean distance metric. Various weighting schemes were tried, with the "best" results obtained by weighting each component variable by the proportion of the variability it explains. The numbers of clusters to form in each case was determined somewhat subjectively by iteratively running the algorithm for various numbers of clusters and examining the sum of the squared deviations about the cluster centroids. The results of the clustering algorithm were illustrated using Kiviat graphs which displayed the approximate fractile ranking of the cluster centroids for each cluster. Kiviat graphs were not originally designed for this purpose. They are useful, however, in presenting the multidimensional nature of workload data.

Two clusters, one for Autobatch and one for Batch, were selected as models to use in the design of synthetic jobs. Following the design of the two jobs, a completely randomized factorial design was used to guide the collection of data and to test the significance of the effects that the synthetic job parameters have on the various resource demands. Regression analysis was performed to yield predictor equations for the resource demands as functions of the synthetic job parameters.

CHAPTER VII

SUMMARY AND CONCLUSIONS

7.1 Review of the Proposed Methodology

The construction of a representative test workload is an integral part of any computer performance evaluation study. A methodology which is proposed for use in constructing test workloads has emerged from this research. The major elements of this methodology are

(a) selecting the workload subset by constructing an overall workload profile and then choosing a period which exhibits characteristics pertinent to the evaluation study,

(b) choosing a set of descriptor variables which is detailed enough to represent the demand placed upon the major system resources, but is not so detailed as to complicate later stages of analysis,

(c) collecting data reflecting the values of the descriptor variables for the worksteps in the selected subset,

(d) scaling the resource demand matrix so that each descriptor variable has mean 0 and variance 1,

(e) applying principal components analysis to the scaled resource demand matrix and retaining only those components needed to explain the major part of the variability in the data,

(f) clustering the transformed resource demand vectors in the principal components space using a non-hierarchical clustering

algorithm with a weighted Euclidean distance measure,

(g) designing synthetic jobs for each of the isolated clusters using regression analysis to obtain predictor equations for the parameter settings,

(h) forming a synthetic job mix by combining a sufficient number of copies of the various synthetic jobs with appropriate parameter settings and the desired arrival time of each, and

(i) validating the generated synthetic job mix by executing it on the system being studied, comparing its characteristics with those of the real workload subset, and adjusting the parameter settings as necessary.

7.2 Automatic Generation of Test Workloads

The construction of test workloads is a time consuming, tedious and error prone procedure. Using the proposed methodology, the major portion of this task can be automated. Automation will release the analyst from this tedious chore. It will also provide benefits in the areas of flexibility, ease of modification, and reproducibility. This section will describe the design of an automatic benchmark generator based upon the proposed methodology.

It is not likely that the first three elements of the proposed methodology can be automated to any degree. Considerable insight is required to select an appropriate workload subset and to determine the set of descriptor variables which will adequately represent a given workstep's true demand on the system. Furthermore, the criteria used to judge a workload subset applicable to a given study changes

from one study to the next. One study may require an I/O bound workload; another study may require a compute bound workload; and a third study may require a balanced workload. It is a straightforward task to collect the appropriate data, once the desired workload subset is selected and the descriptor variables determined. In the remainder of this section, then, it will be assumed that the real workload is presented to the generator in the form of a resource demand matrix. The arrival time of the request, possibly its originating location if operating in a distributed environment, and a flag indicating the type of workstep (i.e. transaction, job) are appended to each resource demand vector.

The characterization phase of the analysis combining scaling, principal components analysis, and clustering can be easily automated. It is envisioned that the various classes of workload requests (i.e. batch, time-sharing, and real-time) would be first segregated. Analysis would proceed separately on the different classes. Some decisions would still need to be made by the analyst. These include how many of the principal components to retain and how many clusters to form if non-hierarchical clustering is used. The first decision on retention of principal components can be built into the generator. That is, it may be decided to retain sufficient components to explain a particular proportion of the variability in all cases. The second decision is not so readily made, since the "optimal" number of clusters to form is largely data dependent. There is the need for a clustering algorithm which does not require this decision.

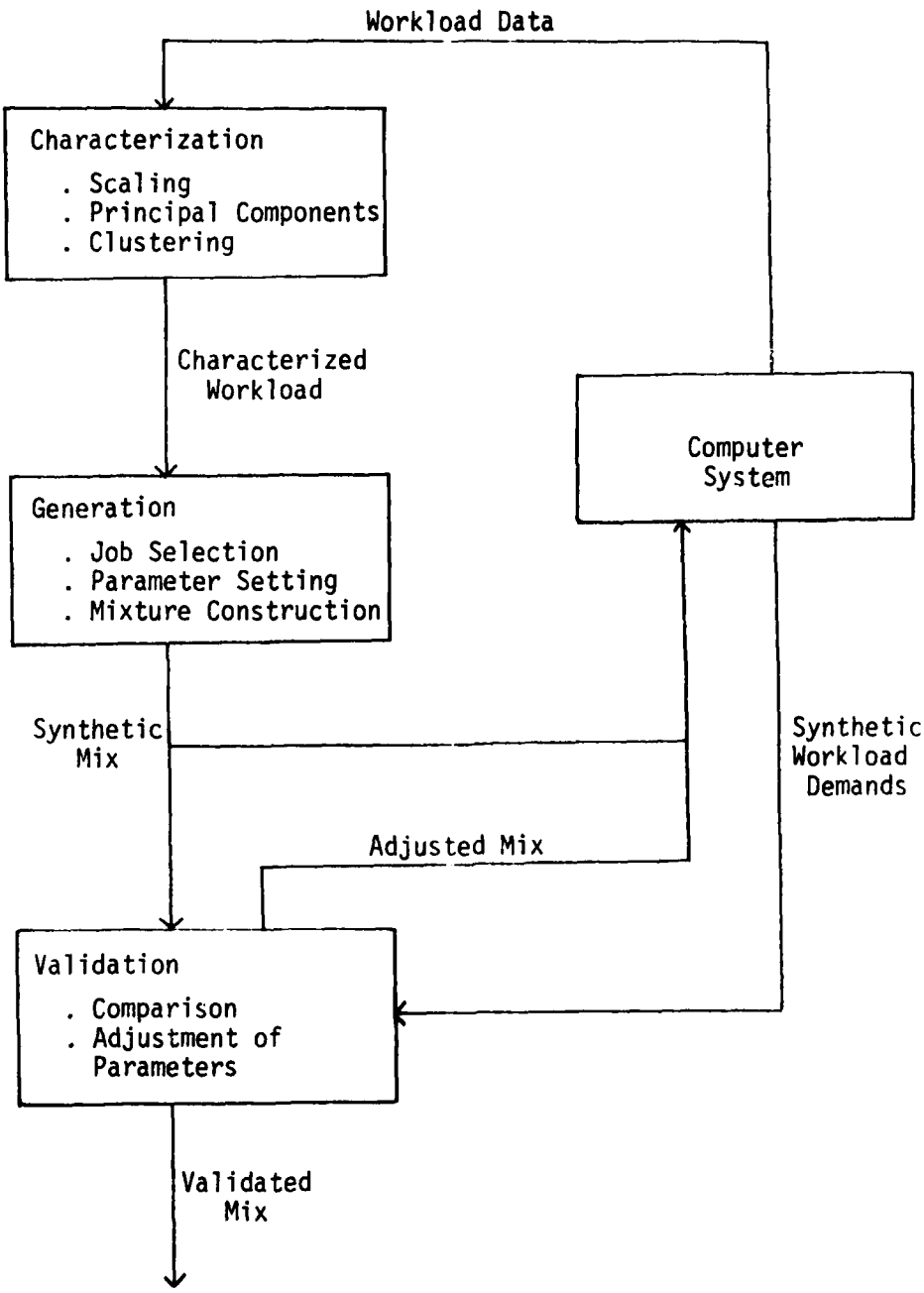
The next two elements of the methodology are also amenable to

automation. It would require the construction of a library of general purpose synthetic jobs. This library must contain synthetic versions of batch processing as well as transaction oriented jobs. The appropriate synthetic job would be selected from this library by first determining the type of job (i.e. batch or interactive) needed by examining the flag appended to the resource demand vector. The required resource demands would then be compared against those demands which could be produced by the various library jobs. The appropriate parameter settings could then be calculated using previously developed predictor equations. Following the selection of library jobs and the determination of the required parameter settings, the synthetic mix could be generated by considering the time and location of origin for each workstep.

Calibration/validation of the produced synthetic job mix is necessary to assure its representativeness. This requires that the synthetic mix be executed on the system, and data collected on the resources used. The resource utilization pattern for the synthetic mix is compared to that of the original workload subset. Parameters are adjusted, and the process repeated until the desired agreement is reached. The details of this procedure are not clear, however it appears feasible.

An automatic benchmark generator then would be composed of three basic modules: a characterization module, a benchmark generator module and a calibration/validation module. These modules are depicted in figure 7.1.

Fig. 7.1 Automatic Benchmark Generator



7.3 Major Points Originated by the Research

This study differs substantially from previous workload characterization studies. These differences are in the following areas:

(a) This study proposes a complete statistical methodology which can be used to construct test workloads. Previous studies were generally restricted to a portion of the problem.

(b) This study separated the workload characterization problem for management oriented studies from that of constructing test workloads. Workload subsets selected at random from a computer workload are not likely to be applicable to the test workload construction problem.

(c) This study examined the intercorrelations among the descriptor variables and their effects on the clustering phase of the analysis. Previous studies have largely ignored this problem.

(d) Principal components analysis was used to reduce the dimensionality of the descriptor space. This is believed to be the first application of this technique to the workload problem, although one report [80] suggested its possible utility. Previous attempts at reducing the dimensionality of the descriptor space have been inconclusive and self-defeating.

(e) Various clustering algorithms and weighting schemes were compared in this research as they apply to the workload problem. Previous studies seemed to rely upon a given scheme with little motivation for its use.

(f) A general purpose synthetic job for use with batch workloads was developed. By varying the parameter settings, this job can perform

as an I/O bound job, a compute bound job, or a balanced job. It includes the facility for tape, disk, and unit record I/O in a somewhat arbitrary proportion.

(h) The appropriate parameter settings for the synthetic jobs were determined from predictor equations obtained through regression analysis. Statistical experimental design techniques were used to guide the collection of data, and to allow testing of the significance of the effect that various parameters have on resource demands. As far as can be determined, these techniques have not previously been applied to this problem although they are routinely applied in other areas.

7.4 Suggested Areas for Future Research

The methodology which has emerged from this research has not been subjected to the test of time. The case study of chapter VI demonstrated the usefulness of many of the procedures employed, however, they need to be applied to other sets of data at other installations to gain a degree of acceptance. A complete, ready to run synthetic benchmark was not produced in the case study due to a need to limit its scope. This needs to be done so that the calibration/validation phase of the procedure can be more clearly defined.

The "best" clustering algorithm found for this study is a non-hierarchical clustering algorithm which requires the analyst to decide how many clusters to form. This decision is somewhat subjective, and is certainly data dependent. There is the need for a clustering algorithm which removes the burden of this decision form the analyst. This is particularly critical if the procedure is to be automated.

Development of such an algorithm would minimize the degree of human intervention in the generation process.

7.5 Conclusions

There is the need for the construction of test workloads for use in computer performance evaluation studies. This research has produced a statistical methodology which should prove useful in this construction process. The feasibility of the major portions of this methodology was demonstrated with a detailed case study of the Amdahl 470/V6 at Texas A&M University.

As with any statistical procedure, there are certain precautions which must go along with the proposed methodology. Two major elements, principal components analysis and clustering, have been the subject of widespread misuse in the past [7]. The problem basically comes from attaching "truth" to the results obtained from these purely mechanical procedures. The results of principal components analysis are scale dependent; the results of clustering are dependent upon the distance metric and weighting scheme used. Both, however, can prove to be effective tools if used in a sound manner [7].

REFERENCES

1. Abrams, M.D., and Cotton, I.W. The Service Concept Applied to Computer Networks. National Bureau of Standards Technical Note 880 (Aug. 1975).

2. Afifi, A.A., and Azen, S.P. Statistical Analysis, A Computer Oriented Approach. Academic Press, New York, 1972.

3. Agrawala, A.K. The Relationship Between the Pattern Recognition Problem and the Workload Characterization Problem. Proc. 1977 SIGMETRICS/CMG VIII Performance Conference, 131-139.

4. Agrawala, A.K., and Bryant, R.M. An Approach to the Workload Characterization Problem. Computer 9, 6 (Jun. 1976), 18-32.

5. Agrawala, A.K., and Mohr, J.M. Some Results on the Clustering Approach to Workload Modelling. Proc. CPEUG, National Bureau of Standards Special Publication 500-18 (Sep. 1977), 23-38.

6. Aho, A.V., Hopcroft, J.E., and Ullman, J.D. The Design and Analysis of Computer Algorithms. Addison-Wesley Publishing Company, Reading, Ma. 1976.

7. Anderberg, M.R. Cluster Analysis for Applications. Academic Press, New York, 1973.

8. Anderson, H.A., and Sargent, R.G. A Statistical Evaluation of the Scheduler of an Experimental Interactive Computing System. Statistical Computer Performance Evaluation (Frieberger, W., Ed.). Academic Press, New York, 1972, 73-98.

9. Bard, Y. Performance Criteria and Measurement for a Time-Sharing System. IBM Sys. J. 10, 4 (1971), 305-324.

10. Bard, Y. The VM/370 Performance Predictor. ACM Computing Surveys 10, 3 (Sep. 1978), 333-342.

11. Barr, A.J., Goodnight, J.H., Sall, J.P., and Helwig, J.T. A User's Guide to SAS 76. SAS Institute, Inc., Raleigh, N.C., 1976.

12. Bear, J.R., and Reeves, T.E. Workload Characterization and Performance Measurement for a CDC CYBER 74 Computer System. Proc. CPEUG, National Bureau of Standards Special Publication 500-18 (Sep. 1977), 39-67.

13. Bell, T.E., Boehm, B.W., and Jeffrey, S. (Editors). Computer Performance Evaluation: Report of the 1973 NBS/ACM Workshop, National Bureau of Standards Special Publication 406 (Sep. 1975).

14. Benwell, N. Benchmarking-Computer Evaluation and Measurement. John Wiley and Sons, New York, 1975.

15. Bonner, A.J. Using System Monitor Output to Improve Performance. IBM Sys. J. 8, 4 (1969), 290-297.

16. Bonner, R.E. On Some Clustering Techniques. IBM J. 8, 1 (Jan. 1964), 22-32.

17. Boyse, J.W., and Warn, D.R. A Straightforward Model for Computer Performance Prediction. ACM Computing Surveys 7, 2 (Jun. 1975), 73-94.

18. Buchholz, W. A Synthetic Job for Measuring System Performance. IBM Sys. J. 8, 4 (1969), 309-318.

19. Buzen, J.P. A Queueing Network Model of MVS. ACM Computing Surveys 10, 3 (Sep. 1978), 319-332.

20. Calingaert, P. System Performance Evaluation: Survey and Appraisal. Comm. ACM 10, 1 (Jan. 1967), 12-18.

21. Chandy, K.M., and Sauer, C.H. Approximate Methods for Analyzing Queueing Network Models of Computer Systems. ACM Computing Surveys 10, 3 (Sep. 1978), 281-318.

22. Cheng, P.S. Trace Driven System Modeling. IBM Sys. J. 8, 4 (1969), 280-289.

23. Coffman, E.G., Jr., and Denning, P.J. Operating Systems Theory. Prentice-Hall, Englewood Cliffs, N.J., 1973.

24. Crothers, E.G. Workload Determination and Representation for Online Computer Systems, ESO-TR-74-54. The Mitre Corporation, Bedford, Massachusetts, NTIS-AD-779 818 (Jan. 1974).

25. Curnow, N.J., and Wichmann, B.A. A Synthetic Benchmark. Computer J. 19, 1 (Feb. 1976), 43-49.

26. Davies, D.J.M. Analysis of Variability in System Accounting Data. Proc. CPEUG, National Bureau of Standards Special Publication 500-41 (Oct. 1978), 243-253.

27. Denning, P.J., and Buzen, J.P. The Operational Analysis of Queueing Network Models. ACM Computing Surveys 10, 3 (Sep. 1978), 225-262.

28. Draper, N.R., and Smith, H. Applied Regression Analysis. John Wiley and Sons, New York, 1966.

29. Drummond, M.E. Evaluation and Measurement Techniques for Digital Computer Systems. Prentice-Hall, Englewood Cliffs, N.J., 1973.

30. Fangmeyer, H., Gloden, R., and Larisse, J. An Automatic Clustering Technique Applied to Workload Analysis and System Tuning. Modelling and Performance Evaluation of Computer Systems (Beilner, H., and Gelenbe, E., Ed.). North-Holland Publishing Company, Amsterdam, 1977.

31. Ferrari, D. Workload Characterization and Selection in Computer Performance Measurement. Computer 5, 4 (Jul./Aug. 1972), 18-24.

32. Ferrari, D. Computer System Performance Evaluation. Prentice-Hall, Englewood Cliffs, N.J., 1978.

33. Flynn, M.J. Trends and Problems in Computer Organizations. Proc. IFIPS Congress 74 (1974), 3-10.

34. Fuller, S.H. Performance of an I/O Channel with Multiple Paging Drums, Proc. First Annual SICME Symposium on Measurement and Evaluation (Feb. 1973), 13-21.

35. Gibson, J.C. The Gibson Mix. IBM Tech. Rept. TR00.2043. IBM T.J. Watson Research Center, Yorktown Heights, N.Y. (Jun. 1970).

36. Gnandesikan, R. Methods for Statistical Data Analysis of Multivariate Observations. John Wiley and Sons, New York, 1977.

37. Graham, G.S. Queueing Network Models of Computer System Performance. ACM Computing Surveys 10, 3 (Sep. 1978), 219-224.

38. Graham, R.M. Performance Prediction. Software Engineering: An Advanced Course. Springer-Verlag, Berlin, 1973.

39. Grenander, U., and Tsao, R.T. Quantitative Methods for Evaluating Computer System PErformance: A Review and Proposals. Statistical Computer Performance Evaluation (Freiberg, W., Ed.). Academic Press, New York, 1972, 3-25.

40. Gross, D., and Harris, C. Fundamentals of Queueing Theory. John Wiley and Sons, New York, 1974.

41. Hansemann, F., Kistler, W., and Schulz, H. Modeling for Computer Center Planning. IBM Sys. J. 10, 4 (1971), 305-324.

42. Hellerman, H., and Conroy, T.F. Computer System Performance. McGraw-Hill, New York, 1975.

43. Hicks, C.R. Fundamental Concepts in the Design of Experiments. Holt, Rinehart and Winston, New York, 1973.

44. Holberton, F.E., and Parker, E.G. NBS FORTRAN Test Programs, National Bureau of Standards Special Publication 399 (Oct. 1974).

45. Hughes, J.H. A Functional Instruction Mix and Some Related Topics. Proc. of the International Symp. on Computer Performance Modeling, Measurement and Evaluation (Mar. 1976) 145-153.

46. Hunt, E., Diehr, G., and Garuatz, D. Who are the Users? - An Analysis of Computer Use in University Computer Centers. AFIPS Proc. SJCC 38 (1971), 231-238.

47. IBM Corp. System Management Facilities (SMF) IBM System 360 Operating System Reference Manual No. GC28-6712. IBM Data Processing Division, White Plains, N.Y. (1971).

48. IBM Corp. OS/VS Service Aids Manual No. GC28-0633-1. IBM Data Processing Division, White Plains, N.Y. (1972).

49. Johnson, L.A. Validation-All Important in Benchmarking. Proc. CPEUG, National Bureau of Standards Special Publication 500-18 (Sep. 1977), 75-83.

50. Johnson, S.C. Hierarchical Clustering Schemes. Psychometrika XXXII (1967), 241-254.

51. Joslin, E.O. Applications Benchmarks: The Key to Meaningful Computer Evaluations. Proc. ACM 20th Nat. Conf. (1965), 22-37.

52. Joslin, E.O. and Aiken, J.J. The Validity of Basing Computer Selection on Benchmark Results. Computers and Automation 15, 1 (Jan. 1966), 22-23.

53. Karush, A.D. Benchmark Analysis of Timesharing Systems AD 689 781. Systems Development Corp., Santa Monica, Ca. (1969).

54. Kendall, M.D. A Course in Multivariate Analysis. Hafner Publishing Company, New York, 1957.

55. Kernighan, B.W., and Hamilton, P.A. Synthetically Generated Performance Test Loads for Operating Systems. Proc. 1st ACM SIGME Symp. on Measurement and Evaluation (Feb. 1973), 121-126.

56. Kimbleton, S.R. Performance Evaluation-A Structured Approach. AFIPS Proc. SJCC 40 (1972), 411-416.

57. Kleinrock, L. A Continuum of Time-Sharing Scheduling Algorithms. AFIPS Proc. SJCC 37 (1970), 453-458.

58. Kleinrock. L. Queueing Systems, Vol. I: Theory. John Wiley and Sons, New York, 1975.

59. Kolence, K.W., and Kiviat, P. Software Unit Profiles and Kiviat Figures. Performance Evaluation Review 2, 3 (Sep. 1973), 2-12.

60. Kumar, B. Performance Evaluation of a Highly Concurrent Computer by Deterministic Simulation. M.S. Thesis, University of Illinois (Urbana-Champaign), 1976.

61. Kumar, B., and Davidson, E.S. Performance Evaluation of Highly Concurrent Computers by Deterministic Simulation. Comm. ACM 21, 11 (Nov. 1978), 904-913.

62. Lasseter, G.L., Chandy, K.M., and Browne, J.C. Statistical Pattern Based Models for CPU Burst Prediction. Proc. Computer Science and Statistics: 7th Annual Symp. on the Interface (Oct. 1973), 123-129.

63. Lindsay, D.S. A Study in Operating System Performance Measurement and Modeling. Ph.D. Thesis, University of California (Berkeley), 1975.

64. Lucas, H.C. Performance Evaluation and Monitoring. ACM Computing Surveys 3, 3 (Sep. 1971), 79-91.

65. Lucas, H.C. Synthetic Program Specifications for Performance Evaluations. Proc. ACM 25th Nat. Conf. (1972), 1041-1058.

66. Mamrak, S.A., and Amer, P.D. A Feature Selection Tool for Workload Characterization. Proc. 1977 SIGMETRICS/CMG VIII Performance Conf., 113-120.

67. Mattson, R.L., Gecsei, J., Slutz, D.R., and Traiger, I.L. Evaluation Techniques for Storage Heirarchies. IBM Sys. J. 9, 2 (1970), 78-117.

68. McKinney, J.M. A Survey of Analytical Time-Sharing Models. ACM Computing Surveys 1, 2 (Jun. 1969), 105-116.

69. Merril, H.E.B. A Technique for Comparative Evaluation of Kiviat Graphs. Performance Evaluation Review 4, 1 (Jan. 1975), 1-10.

70. Mohr, J., and Agrawala, A. A Markovian Model of a Job. Proc. CPEUG, National Bureau of Standards Special Publication 500-41 (Oct 1978), 119-128.

71. Morris, F.M. Kiviat Graphs-Conventions and Figures of Merit. Performance Evaluation Review 3, 3 (Oct. 1974), 2-8.

72. Morrison, D.F. Multivariate Statistical Methods. McGraw-Hill, New York, 1976.

73. Noble, B., and Daniel, J.W. Applied Linear Algebra. Prentice-Hall, Englewood Cliffs, N.J., 1977.

74. Nolan, L.E., and Strauss, J.C. Workload Characterization for Timesharing System Selection. Software-Practice and Experience 4, 1 (Jan.-Mar. 1974) 25-39.

75. Nutt, G.J. Tutorial: Computer System Monitors. Computer 8, 11 (Nov. 1975), 51-61.

76. Oliver, P., Baird, G., Cook, M., Johnson, A., and Hoyt, P. An Experiment in the Use of Synthetic Programs for System Benchmarks. AFIPS Proc. NCC 43 (1974), 431-438.

77. Rose, C.A. A Measurement Procedure for Queueing Network Models of Computer Systems. ACM Computing Surveys 10, 3(Sep. 1978), 263-280.

78. Rosen, S. Lectures on the Measurement and Evaluation of the Performance of Computing Systems. SIAM, Philadelphia, 1976.

79. Scherr, A.L. An Analysis of Time-shared Computer Systems, MIT Press, Cambridge, Ma., 1967.

80. Schroeder, A. How Multidimensional Data Analysis can be of Help in the Study of Computer Systems. Proc. CPEUG, National Bureau of Standards Special Publication 500-41 (Oct. 1978), 149-165.

81. Schwetman, H.D., and Browne, J.C. An Experimental Study of Computer System Performance. Proc. 25th ACM Nat. Conf. (1972).

82. Shemer, J.E., and Robertson, J.B. Instrumentation of Time-shared Systems. Computer 5, 4 (Jul./Aug. 1972), 39-48.

83. Shope, W.L., Kashmark, K.L., Inghram, J.W., and Decker, W.R. System Performance Study. Proc. SHARE XXXIV 1 (1970), 439-530.

84. Smith, A.J. A Performance Analysis of Multiple Channel Controllers. Proc. 1st ACM SIGME Symp. on Measurement and Evaluation (Feb. 1973), 37-46.

85. Spirn, J.R. Program Behavior: Models and Measurements. Elsevier North-Holland, New York, 1977.

86. Sreenivasan, K., and Kleinman, A.J. On the Construction of a Representative Synthetic Workload. Comm. ACM 17, 3 (Mar. 1974), 127-133.

87. Strauss, J.C. A Benchmark Study. AFIPS Proc. FJCC 41 (1972), 1225-1233.

88. Svobodova, L. Computer Performance Measurement and Evaluation Methods: Analysis and Applications. American Elsevier, New York, 1976.

89. Timm, N.H. Multivariate Analysis with Applications in Education and Psychology. Wadsworth Publishing Company, Belmont, Ca., 1975.

90. Winder, R.O. A Data Base for Computer Performance Evaluation. Computer 6, 3 (Mar. 1973), 25-29.

91. Wood, D.C., and Forman, E.H. Throughput Measurement Using a Synthetic Job Stream. AFIPS Proc. FJCC 39 (1971), 51-56.

92. Wright, L.S., and Burnette, W.R. An Approach to Evaluating Time Sharing Systems: MH-TSS A Case Study. Performance Evaluation Review (Jan. 1976), 8-28.

APPENDIX A

This appendix describes a computational procedure for representing an mxn data matrix $\vec{X}$ in terms of its principal components. This procedure was utilized to express the scaled resource demand matrix in terms of uncorrelated variables to preclude the biasing of clustering results, and to allow a reduction in the dimensionality of the data matrix as a prelude to clustering.

Let $\vec{X} = \{X_{ij}\}$ be an mxn data matrix, where X_{ij} represents the value of the $j^{\underline{th}}$ variable for the $i^{\underline{th}}$ data unit. Since, at least in the workload characterization problem, the variables are expressed in widely differing units, the data must be scaled to commensurable ranges. Assume that the elements of $\vec{X}$ have been standardized so that each variable represented has mean 0, variance 1.

The variance-covariance matrix for the scaled data matrix $\vec{X}$ is given by $\vec{S} = \frac{\vec{X}^T\vec{X}}{m} = \{S_{ij}\}$. Since the elements of $\vec{X}$ were standardized, $\vec{S}$ is the correlation matrix of the original variables in $\vec{X}$.

Now, define a new variable Y_1 as

$$Y_1 = \sum_{i=1}^{n} B_i X_i \text{ , where the } X_i, \; i=1, \ldots, n,$$

are the original variables, and B_i, i=1, ..., n, are coefficients to be determined. The row vector of coefficients $\vec{B}$ could be defined in a

number of ways, however the principal component solution requires that the variance of Y_1 be maximal [7]. If the data matrix is evaluated in terms of the new variable Y_1, the column vector $\vec{Y}_1 = (Y_{11}Y_{21}Y_{31}\cdots Y_{m1})^T$ given by $\vec{Y}_1 = \vec{X}\vec{B}^T$ would result. Then

$$\text{Var}(Y_1) = \frac{\vec{Y}_1^T\vec{Y}_1}{m} = \frac{\vec{B}\vec{X}^T\vec{X}\vec{B}^T}{m} = \vec{B}\vec{S}\vec{B}^T.$$

By choosing the elements of $\vec{B}$ large, $\text{Var}(Y_1)$ could be made as large as desired. Generally, the convention that $\vec{B}\vec{B}^T = 1$ (i.e. $\vec{B}$ is of unit length) is adopted. This constraint can be linked to the objective function using a Lagrange multiplier μ. Then, a value for $\vec{B}$ which yields maximal variance for Y_1 is found by differentiating with respect to $\vec{B}$ and setting this derivative equal to zero. Thus

$$\frac{d}{d\vec{B}}\left[\vec{B}\vec{S}\vec{B}^T + \mu(1 - \vec{B}\vec{B}^T)\right] = 2\vec{S}\vec{B}^T - 2\mu\vec{B}^T = 0$$

To yield maximal variance for Y_1, one must choose the vector $\vec{B}$ to satisfy $[\vec{S} - \mu\vec{I}]\,\vec{B}^T = 0$. This is an ordinary eigenproblem. Then, the vector $\vec{B}^T$ is one of the eigenvectors of the matrix $\vec{S}$. It is easily shown [7] that in this case, $\vec{B}^T$ is the eigenvector corresponding to the largest eigenvalue λ_1 of $\vec{S}$. The variable Y_1 thus selected is called the first principal component of $\vec{X}$.

Using a procedure similar to the above, it can be shown that the second principal component of $\vec{X}$ is produced using the eigenvector (selected orthogonal to $\vec{B}^T$ above) corresponding to the second largest eigenvalue of $\vec{S}$. Likewise, the third, fourth, ..., $n^{\underline{th}}$ principal components are obtained using eigenvectors associated with the third, fourth, ..., $n^{\underline{th}}$ largest eigenvalues of $\vec{S}$.

Once all principal components have been determined, a matrix of

principal component scores can be computed by the matrix equation $\vec{Y} = \vec{X}\vec{P}$, where $\vec{Y}$ is the matrix of component scores, $\vec{X}$ is the standardized scores, and $\vec{P}$ is a matrix of coefficients formed by placing each of the eigenvectors determined above as a column in $\vec{P}$ (the vector corresponding to the first principal component is the first column, etc.).

Since calculation of principal components is basically an eigenproblem, it is easily attacked using standard matrix manipulation software available at most computer installations. The facilities provided by the Statistical Analysis System [11] were used to isolate the principal components and compute the component scores for the workload data analyzed in Chapter VI.

APPENDIX B

This appendix details the clustering algorithm used in summarizing the real workload subset. The algorithm is the convergent k-means approach discussed by Anderberg [7], and the program developed is modeled after source listings contained in that reference.

The convergent k-means approach involves three basic steps [7]. These are:

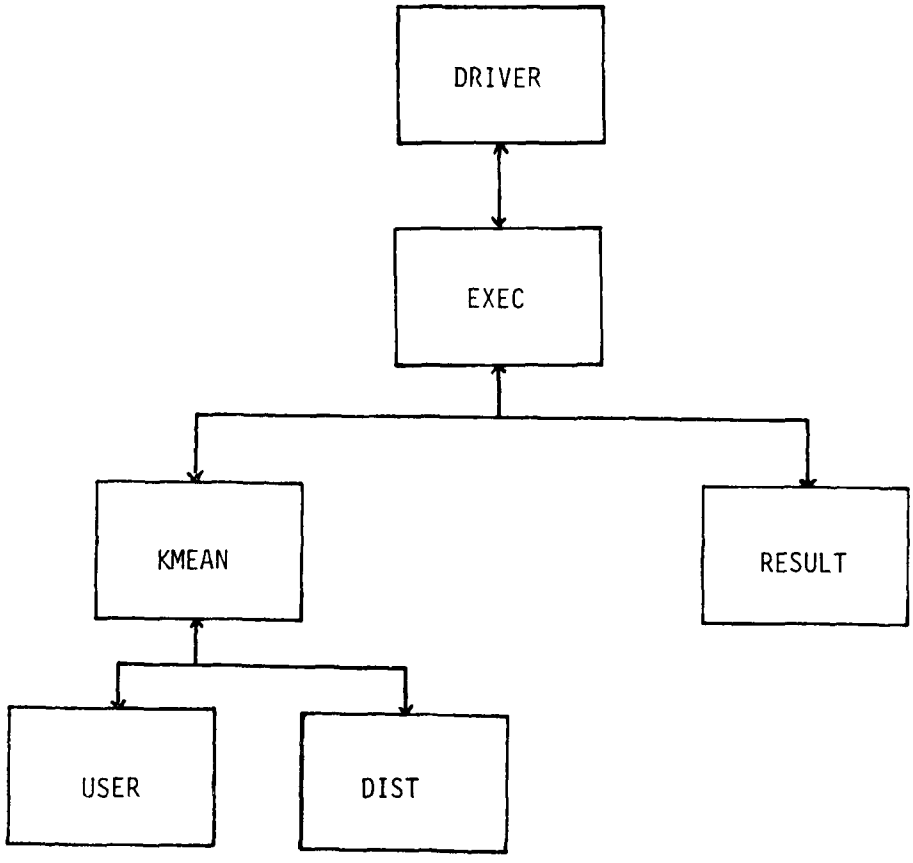
(a) Begin with an initial partition of the data units into clusters. This initial partition can be arrived at in a variety of ways. One way is to select k of the data units as cluster centroids. These k units can be selected at random, the first k units of the data set used, or some other technique employed. The remainder of the data units are then assigned to the "nearest" cluster, with the cluster centroid remaining fixed throughout the initial pass through the data. Once all data units are assigned to a cluster, the centroid vectors are updated to reflect the current cluster memberships.

(b) Take each data unit in sequence, compute the distances to all cluster centroids, and reallocate the data unit if its parent cluster is not the "nearest" cluster. In the event of reallocation, the centroids of both the gaining and losing clusters are updated.

(c) Repeat step (b) until a full pass is made through the data set with no reallocation of data units among clusters.

The convergent k-means algorithm described by Anderberg and implemented for this study consists of a main program (driver) and five subprograms. The logical relations among the elements are depicted in figure B.1.

Fig. B.1 Logical Program Linkages



The main program (DRIVER) simply assigns main storage, and then invokes subroutine EXEC. This subroutine checks that sufficient main storage has been requested and then invokes subroutines KMEAN and RESULT in turn. Subroutine KMEAN is the heart of the algorithm. The

data units are read in, standardized and expressed in terms of principal component scores through repeated calls to subroutine USER. Clustering is then accomplished with distance measures between data units and cluster centroids obtained through invocation of function DIST. Once clustering is achieved, subroutine RESULT is called to output the results.

There are a number of decisions which must be made by the analyst prior to using this algorithm. These include how the initial partition is arrived at, how many clusters are formed, and what measure of distance is used.

For this study, the first k data units were used as the "seeds" of the algorithm. This choice was made for lack of a decidedly better alternative. Some experimentation with other techniques was done, however, the results did not consistently favor one over the other. Thus, the easiest and most straightforward approach was taken.

The particular implementation of the clustering algorithm used in this study is shown in the following source listings. The listings contain liberal comments on the different logical stages, rendering further explanation unnecessary.

Fig. B.2 Clustering Algorithm - Program Listings

```
C
C     THIS DRIVER ACCOMPLISHES TWO TASKS:
C       1. ESTABLISHES MAIN STORAGE REQUIREMENTS.
C       2. CALLS SUBROUTINE EXEC.
C
      DIMENSION X(5000)
      LIMIT=5000
      CALL EXEC(X,LIMIT)
      END
      SUBROUTINE EXEC(X,LIMIT)
C
C     THIS SUBROUTINE READS PARAMETERS, COMPUTES STORAGE LOCATIONS
C     AND CALLS MAJOR PROGRAM SEGMENTS NEEDED TO PERFORM NON-
C     HEIRARCHICAL CLUSTERING.
C
C     INPUT SPECIFICATIONS:
C     CARD 1 - TITLE
C     CARD 2 - PARAMETER CARD
C       COLS 1-5   NE=NUMBER OF ENTRIES(DATA UNITS)
C       COLS 6-10  NV=NUMBER OF VARIABLES
C       COLS 11-15 NC=NUMBER OF CLUSTERS TO BE FORMED
C       COLS 16-20 NTIN=INPUT UNIT FOR DATA UNITS
C       COLS 21-25 NTOUT=OUTPUT UNIT FOR SAVING MEMBERSHIP LISTS
C                       (NTOUT=0, DO NOT SAVE)
C       COLS 26-30 MINREL=TERMINATION PARAMETER
C                        (MINREL=0,ITERATE TO COMPLETE CONVERGENCE)
C       COLS 31-35 IPART=INITIAL PARTITION PARAMETER
C                       IPART=1 SEED POINTS ARE SELECTED FROM THE
C                               DATA UNITS.READ THE SEQUENCE NRS
C                               FROM CARD 3 IN 2014 FORMAT.
C                       IPART=2 THE DATA UNITS ARE GROUPED INTO
C                               AN INITIAL PARTITION WITH FIRST
C                               NUMBR(1) IN PARTITION 1,ETC.READ
C                               THE NUMBR ARRAY FROM CARD 3 IN
C                               2014 FORMAT.
```

Fig. B.2 continued

```
C                          IPART=3 THE SCORE VECTORS FOR THE INITIAL
C                                  PARTITIONS ARE READ FROM CARD 4
C                                  IN FORMAT FMT WHICH IS READ FROM
C                                  CARD 3.
C
C     STORAGE ALLOCATIONS IN THE X ARRAY:
C       X(N1)  TO X(N2-1)  NC*NV WORDS - STORAGE OF CENTR ARRAY
C       X(N2)  TO X(N3-1)  NC WORDS - STORAGE OF NUMBR ARRAY
C       X(N3)  TO X(N4-1)  NE WORDS - STORAGE OF MEMBR ARRAY
C       X(N4)  TO X(N5-1)  NC*NV WORDS - STORAGE OF TOTAL ARRAY
C       X(N5)  TO X(N6)    NV OR NE*NV WORDS - STORAGE OF DATA ARRAY
C       X(N4)  TO X(N7)    NE WORDS - STORAGE OF LIST ARRAY
C
      DIMENSION X(1),TITLE(20)
      READ(5,1000) TITLE
      READ(5,1100) NE,NV,NC,NTIN,NTOUT,MINREL,IPART
      WRITE(6,2000)TITLE
      WRITE(6,2100)NE,NV,NC,NTIN,NTOUT,MINREL,IPART
C
C     COMPUTE STORAGE REQUIREMENTS
C
      N1=1
      N2=N1+NC*NV
      N3=N2+NC
      N4=N3+NE
      N5=N4+NC*NV
      N6=N5+NV-1
      N7=N4+NE-1
      MAX=N6
      IF(N7.GT.MAX)MAX=N7
      WRITE(6,2200)MAX,LIMIT
      IF(MAX.GT.LIMIT)STOP
      CALL KMEAN(X(N1),X(N2),X(N3),X(N4),X(N5),N5,NE,NV,NC,NTIN,
     *MINREL,IPART,LIMIT)
      CALL RESULT(X(N1),X(N2),X(N3),X(N4),TITLE,NE,NV,NC,NTOUT)
```

Fig. B.2 continued

```
      RETURN
1000  FORMAT(20A4)
1100  FORMAT(7I5)
2000  FORMAT(1H1,20A4)
2100  FORMAT(5H0NE =,I8,/,5H NV =,I8,/,5H NC =,I8,/,7H NTIN =,I6,/,
     *8H NTOUT =,I5,/,9H MINREL =,I4,/,8H IPART =,I5)
2200  FORMAT(19H0REQUIRED STORAGE =,I5,6H WORDS,/,
     *       19H0ALLOTTED STORAGE =,I5,6H WORDS)
      END
      SUBROUTINE KMEAN(CENTR,NUMBR,MEMBR,TOTAL,DATA,N5,NE,NV,NC,
     *NTIN,MINREL,IPART,LIMIT)
C
C     THIS SUBROUTINE ITERATIVELY SORTS NE DATA UNITS INTO NC
C     CLUSTERS USING THE CONVERGENT KMEANS METHOD DESCRIBED IN
C     SECTION 7.2.2 OF ANDERBERG,M.,CLUSTER ANALYSIS FOR APPLI-
C     CATIONS,ACADEMIC PRESS,1973.
C
C     CENTR(NV*(J-1)+I)=SCORE ON THE ITH VARIABLE FOR THE JTH
C                       CLUSTER CENTROID.
C     TOTAL(NV*(J-1)+I)=TOTAL SCORE ON THE ITH VARIABLE FOR DATA
C                       UNITS THUS FAR ALLOCATION TO JTH CLUSTER.
C     NUMBR(J)=NUMBER OF DATA UNITS THUS FAR ALLOCATED TO THE
C              JTH CLUSTER.
C     MEMBR(K)=CLUSTER TO WHICH THE KTH DATA UNIT CURRENTLY BELONGS.
C     DATA(NV*(K-1)+I)=SCORE ON THE ITH VARIABLE FOR THE KTH
C                      DATA UNIT
C
      DIMENSION CENTR(1),TOTAL(1),NUMBR(1),MEMBR(1),DATA(1),FMT(20)
      WRITE(6,2000)
C
C     CHECK FOR SUFFICIENT STORAGE
C
      N6=N5+NE*NV-1
      WRITE(6,2100) N6,LIMIT
      IF(N6.GT.LIMIT)STOP
```

Fig. B.2 continued

```
C
C       ESTABLISH INITIAL PARTITION
C
        IF(IPART.NE.3)GO TO 20
C
C       SEED POINTS ARE READ DIRECTLY FROM CARDS
C
        READ(5,1000)FMT
        WRITE(6,2200)FMT
        WRITE(6,2300)
        J1=0
        DO 10 J=1,NC
        READ(5,FMT)(CENTR(J1+I),I=1,NV)
        WRITE(6,2400)(CENTR(J1+I),I=1,NV)
10      J1=J1+NV
        GO TO 30
C
C       IPART=1 OR 2
C
20      WRITE(6,2500)IPART
        READ(5,1100)(NUMBR(J),J=1,NC)
        WRITE(6,2600)(NUMBR(J),J=1,NC)
C
C       READ THE DATA SET INTO MEMORY
C
30      K1=1
        DO 40 K=1,NE
        CALL USER(DATA(K1))
40      K1=K1+NV
        IF(IPART.EQ.3)GO TO 51
C
C       IF IPART IS 1 OR 2 SET UP THE SEED POINTS
C
        IF(IPART.EQ.2)GO TO 60
C
```

Fig. B.2 continued

```
C       IPART=1, THE DATA UNIT WITH SEQUENCE NUMBER NUMBR(J) IS
C                USED AS THE JTH SEED POINT.
C
        DO 50 J=1,NC
        NJ=(NUMBR(J)-1)*NV
        J1=(J-1)*NV
        DO 50 I=1,NV
        CENTR(J1+I)=DATA(NJ+I)
50      CONTINUE
C
C       THE INITIAL CONFIGURATION IS GIVEN IN TERMS OF SEED POINTS.
C       CONSTRUCT AN INITIAL PARTITION BY ASSIGNING EACH DATA UNIT
C       TO THE NEAREST SEED POINT. SEED POINTS REMAIN FIXED
C       THROUGHOUT ASSIGNMENT OF THE FULL DATA SET.
C
51      DO 52 K=1,NE
52      MEMBR(K)=0
        J1=0
        DO 53 J=1,NC
        NUMBR(J)=0
        DO 53 I=1,NV
        J1=J1+1
53      TOTAL(J1)=0.
C
C       ALLOCATE EACH DATA UNIT TO THE CLOSEST SEED POINT
C
        K1=0
        DO 55 K=1,NE
        K2=K1+1
        J2=1
C
C       COMPUTE DISTANCE TO THE FIRST SEED POINT
C
        DREF=DIST(DATA(K2),CENTR(J2))
        JREF=1
```

Fig. B.2 continued

```
C
C       TEST DISTANCES TO REMAINING SEED POINTS
C
        DO 54 J=2,NC
        J2=J2+NV
        DTEST=DIST(DATA(K2),CENTR(J2))
        IF(DTEST.GE.DREF)GO TO 54
        DREF=DTEST
        JREF=J
54      CONTINUE
C
C       ALLOCATE DATA UNIT K TO CLUSTER JREF
C
        NUMBR(JREF)=NUMBR(JREF)+1
        MEMBR(K)=JREF
        J1=(JREF-1)*NV
        DO 55 I=1,NV
        J1=J1+1
        K1=K1+1
        TOTAL(J1)=TOTAL(J1)+DATA(K1)
55      CONTINUE
        GO TO 85
C
C       IPART=2. THE DATA UNITS ARE GROUPED INTO CLUSTERS WITH THE
C                JTH CLUSTER HAVING NUMBR(J) MEMBERS.
60      K=0
        J1=-NV
C
C       ACCUMULATE THE TOTAL SCORE ON EACH VARIABLE FOR EACH CLUSTER
C
        DO 80 J=1,NC
        NJ=NUMBR(J)
        J1=J1+NV
        DO 70 I=1,NV
70      TOTAL(J1+I)=0.
```

Fig. B.2 continued

```
            DO 80 KJ=1,NJ
            K=KJ+1
            MEMBR(K)=J
            K1=(K-1)*NV
            DO 80 I=1,NV
            J2=J1+I
            TOTAL(J2)=TOTAL(J2)+DATA(K1+I)
   80       CONTINUE
   C
   C        COMPUTE THE CENTROIDS
   C
   85       J1=0
            DO 90 J=1,NC
            DO 90 I=1,NV
            J1=J1+1
            CENTR(J1)=TOTAL(J1)/NUMBR(J)
   90       CONTINUE
   C
   C        INITIALIZE ARRAYS
   C
   100      NPASS=1
   C
   C        BEGINNING OF MAIN LOOP
   C
   120      MOVES=0
            TDIST=0
   C
   C        ALLOCATE EACH DATA UNIT TO THE NEAREST CLUSTER CENTROID
   C
            K1=0
            DO 160 K=1,NE
            K2=K1+1
            J2=1
   C
   C        COMPUTE DISTANCE TO THE FIRST CLUSTER CENTROID
```

Fig. B.2 continued

```
C
      DREF=DIST(DATA(K2),CENTR(J2))
      JREF=1
C
C     TEST DISTANCES TO REMAINING CLUSTER CENTROIDS
C
      DO 140 J=2,NC
      J2=J2+NV
      DTEST=DIST(DATA(K2),CENTR(J2))
      IF(DTEST.GE.DREF)GO TO 140
      DREF=DTEST
      JREF=J
140   CONTINUE
      TDIST=TDIST+DREF
      IF(JREF.NE.MEMBR(K))GO TO 155
      K1=K1+NV
      GO TO 160
C
C     REALLOCATE DATA UNIT K FROM CLUSTER MEMBR(K) TO CLUSTER JREF
C
155   MOVES=MOVES+1
      J2=MEMBR(K)
      NUMBR(J2)=NUMBR(J2)-1
      NUMBR(JREF)=NUMBR(JREF)+1
      MEMBR(K)=JREF
      J1=(JREF-1)*NV
      J3=(J2-1)*NV
      DO 150 I=1,NV
      J1=J1+1
      J3=J3+1
      K1=K1+1
      TOTAL(J1)=TOTAL(J1)+DATA(K1)
      CENTR(J1)=TOTAL(J1)/NUMBR(JREF)
      TOTAL(J3)=TOTAL(J3)-DATA(K1)
      CENTR(J3)=TOTAL(J3)/NUMBR(J2)
```

Fig. B.2 continued

```
  150   CONTINUE
  160   CONTINUE
        C
        C       ALL DATA UNITS ALLOCATED - TEST FOR CONVERGENCE
        C
              WRITE(6,2700) MOVES,NPASS,TDIST
              NPASS=NPASS+1
              IF(MOVES.LE.MINREL) RETURN
              GO TO 120
  1000  FORMAT(20A4)
  1100  FORMAT(20I4)
  2000  FORMAT(46H0CONVERGENT K-MEANS METHOD OF CLUSTER ANALYSIS,/,
             * 24H DATA SET STORED IN CORE)
  2100  FORMAT(19H0REQUIRED STORAGE =,I5,6H WORDS,/,
             *       19H0ALLOTTED STORAGE =,I5,6H WORDS)
  2200  FORMAT(7H0FORMAT,20A4)
  2300  FORMAT(43H1INITIAL CLUSTER CENTERS READ IN AS FOLLOWS///)
  2400  FORMAT(1X,10E12.4)
  2500  FORMAT(9H1 IPART =,I2,30H,  NUMBR ARRAY READ AS FOLLOWS///)
  2600  FORMAT(1X,10I7)
  2700  FORMAT(1H0,I5,37H DATA UNITS MOVED ON ITERATION NUMBER,I3,/,
             *38H SUMMED DEVIATIONS ABOUT SEED POINTS=,E16.8)
              END
              SUBROUTINE RESULT(CENTR,NUMBR,MEMBR,LIST,TITLE,NE,NV,NC,NTOUT)
        C
        C       THIS SUBROUTINE PRINTS THE RESULTS OF NON-HEIRARCHICAL
        C       CLUSTERING PRODUCED BY SUBROUTINE KMEAN.
        C
              DIMENSION CENTR(1),NUMBR(1),MEMBR(1),LIST(1),TITLE(20)
        C
        C       WRITE OUT THE RAW MEMBERSHIP LIST AS A PRECAUTION
        C
              WRITE(6,2000)  TITLE
              WRITE(6,2100) (MEMBR(K),K=1,NE)
              WRITE(6,2200) (NUMBR(J),J=1,NC)
```

Fig. B.2 continued

```
C
C       INVERT THE MEMBR ARRAY AND PUT THE RESULT IN THE LIST ARRAY
C       FIRST REVISE THE NUMBR ARRAY TO CONTAIN START POINTS IN THE
C       LIST ARRAY FOR EACH CLUSTER.
C
        NUMBR(NC)=NE-NUMBR(NC)+1
        JJ=NC
        JJ1=JJ-1
        DO 10 J=2,NC
        NUMBR(JJ1)=NUMBR(JJ)-NUMBR(JJ1)
        JJ=JJ1
10      JJ1=JJ-1
C
C       BUILD LIST ARRAY
C
        DO 20 K=1,NE
        MEMBRK=MEMBR(K)
        NJ=NUMBR(MEMBRK)
        LIST(NJ)=K
        NUMBR(MEMBRK)=NUMBR(MEMBRK)+1
20      CONTINUE
C
C       SAVE THE MEMBERSHIP LIST IF DESIRED
C
        IF(NTOUT.EQ.0)GO TO 30
        WRITE(NTOUT,3000)TITLE
        WRITE(NTOUT,3100)(LIST(K),K=1,NE)
C
C       RESTORE THE NUMBR ARRAY
C
30      JJ=NC
        DO 40 J=2,NC
        NUMBR(JJ)=NUMBR(JJ)-NUMBR(JJ-1)
40      JJ=JJ-1
        NUMBR(1)=NUMBR(1)-1
```

Fig. B.2 continued

```
C
C       PRINT RESULTS FOR EACH CLUSTER
C
        WRITE(6,2000)TITLE
        K1=1
        DO 50 J=1,NC
        WRITE(6,2300)J,NUMBR(J)
        J1=(J-1)*NV
        WRITE(6,2400)(CENTR(J1+I),I=1,NV)
        K2=K1+NUMBR(J)-1
        WRITE(6,2500)(LIST(K),K=K1,K2)
        K1=K2+1
50      CONTINUE
        RETURN
2000    FORMAT(1H1,20A4)
2100    FORMAT(20HORAW MEMBERSHIP LIST,/,(1X,25I5))
2200    FORMAT(14HOCLUSTER SIZES,/,(1X,25I5))
2300    FORMAT(8HOCLUSTER,I3,9H CONTAINS,I5,11H DATA UNITS)
2400    FORMAT(21HOCENTROID COORDINATES,/,(1X,10E12.4))
2500    FORMAT(16HOMEMBERSHIP LIST,/,(1X,25I5))
3000    FORMAT(20A4)
3100    FORMAT(20I4)
        END
        SUBROUTINE USER(X)
        DIMENSION X(1),IVAR(12),S(12),VMEAN(12),VSTD(12)
        DATA VMEAN/1.560,12.179,159.238,257.869,1872.700,76.107,
       *           31.732,12.435,336.393,812.399,354.673,1009.780/
        DATA VSTD/1.031,10.680,80.769,655.329,4771.510,674.565,
       *          49.158,34.585,1193.590,2948.550,2612.710,4556.950/
C
C       THIS SUBROUTINE READS THE DATA UNITS IN,STANDARDIZES THEM,
C       AND CONSTRUCTS THE PRINCIPAL COMPONENT SCORES.
C
        READ(5,1000)IDNO,(IVAR(I),I=1,10),IHASP,IVAR(11),IVAR(12)
C
```

Fig. B.2 continued

```
C       STANDARDIZE THE VARIABLES
C
        DO 10 I=1,3
        SVAR(I)=(FLOAT(IVAR(I))-VMEAN(I))/VSTD(I)
10      CONTINUE
C
C       CONSTRUCT PRINCIPAL COMPONENTS
C
        X(1)=0.55052*SVAR(1)+0.60964*SVAR(2)+0.57032*SVAR(3)
        X(2)=0.76719*SVAR(1)-0.10011*SVAR(2)-0.63356*SVAR(3)
        X(3)=0.32915*SVAR(1)-0.78633*SVAR(2)+0.52282*SVAR(3)
        RETURN

1000    FORMAT(I3,I5,I5,I4)
        END


        FUNCTION DIST(X,Y)
        DIMENSION X(1),Y(1),W(3)
        DATA W/2.337483,0.457655,0.204861/
C
C       THIS FUNCTION COMPUTES THE DISTANCE BETWEEN TWO VECTORS X AND Y
C       USING THE WEIGHTED EUCLIDEAN DISTANCE MEASURE.
C
        SQDIS=0.
        DO 10 I=1,3
10      SQDIS=SQDIS+W(I)*((X(I)-Y(I))**2)
        DIST=SQRT(SQDIS)
        RETURN
        END
```

APPENDIX C

This appendix describes data collection using the IBM System Management Facility (SMF) as it is implemented on the Amdahl 470/V6 at Texas A&M University. The basic flow of information to SMF is described, and the particular SMF records which were used in this study are detailed.

SMF is an optional feature of the IBM System 360/370 operating systems that can be selected at system generation (SYSGEN) time. SMF collects system, job management, and data management information, and can be linked to user-written routines which can monitor the operation of jobs or job steps [47]. The information is collected for use by management and systems analysts in billing customers or evaluating system usage.

There is a variety of types of information collected by SMF. They include

(a) accounting information such as CPU time and device and storage utilization;

(b) data set activity such as a count of block transfer requests (EXCPs) and the particular user of the data set;

(c) Volume information such as the space available on direct access volumes and error statistics on tape volumes;

(d) system use information such as system wait time and I/O

configuration.

The type of data which is collected can be modified by the operator at each initial program load [IPL]. For example, data set activity is not presently collected at Texas A&M University.

There are a number of different records written by SMF. The original manual [47] listed thirty-one such records. Depending upon the system configuration, some additional records may be added. For example, a HASP purge record reflecting each job's characteristics as viewed by the spooling program and a record monitoring the activity of the WYLBUR/370 system have been added to the collection of SMF records used at Texas A&M University.

The various SMF records are written to the primary SMF data set (SYS1.MANX) at critical points in the lifetime of a job. For example, the job termination record is written whenever the job is terminated either normally or abnormally, and data set information is recorded whenever a data set opened by a user program is scratched, renamed, closed or processed by end-of-volume (EOV). If SYS1.MANX is defined on a direct-access device, as it is at Texas A&M University, an additional SMF data set, SYS1.MANY, is also defined. Data is recorded on SYS1.MANX until its defined extent is reached. At that time, recording is switched to SYS1.MANY, and SYS1.MANX is copied to a dump data set (magnetic tape). Periodically, the dump data sets are merged to provide a complete record of system activity over some period of time (i.e. one month).

The monthly SMF files provide a rich source of resource demand

information which can be used for workload characterization studies. The particular SMF records which were used in this study are detailed in the following tables. They are included here not only for completeness, but also to point out the wide range of workload descriptors which is available, at least on IBM compatible equipment, without recourse to monitor data.

Table C.1 Type 4 (Step End) Record [47]

Decimal Displacement	Field Size	Contents
0	1	Reserved (zero)
1	1	Record type (4)
2	4	Time of end of step
6	4	Date of end of step
10	2	System identification
12	2	System model identifier
14	8	Job name
22	4	Reader start time
26	4	Reader start date
30	8	User identification
38	1	Step number
39	4	Step initiation time
43	4	Step initiation date
47	4	Number of card image records in input data set
51	2	Step completion code
53	1	Step priority
54	8	Program name
62	8	Name of executed step
70	2	Region size in heirarchy 0
72	2	Region size in heirarchy 1
74	4	Storage used in heirarchy 0
78	4	Storage used in heirarchy 1
82	1	Storage protect key
83	3	Reserved
86	4	Device allocation time
90	4	Problem program load time
94	8	Reserved
*102	variable	Devices used by step
variable	1	Total length of next fields
variable	3	Step CPU time
variable	1	No. of accounting fields
variable	variable	Accounting fields

* - Bytes 0 and 1 contain the length of the field. For each assigned device there is an eight byte field giving the device class, unit type, channel and unit address, and a count of the EXCPs issued for the device.

Table C.2 Type 5 (Job Termination) Record [47]

Decimal Displacement	Field Size	Contents
0	1	Reserved (zero)
1	1	Record type (5)
2	4	Time of end of job
6	4	Date of end of job
10	2	System identification
12	2	System model identifier
14	8	Job name
22	4	Reader start time
26	4	Reader start date
30	8	User identification
38	1	Number of steps in job
39	4	Job initation time
43	4	Job initiation date
47	4	Number of card images in input data set
51	2	Job completion code
53	1	Job priority
54	4	Reader stop time
58	4	Reader stop date
62	1	Job termination indicator
63	5	Output class indicator
68	1	Checkpoint/restart indicator
69	1	Reader device class
70	1	Reader unit type
71	1	Job input class
72	1	Storage protect key
73	19	Reserved
92	1	Length of rest of record
93	20	Programmer's name
113	3	CPU time for job
116	1	Number of accounting fields
117	variable	Accounting fields

Table C.3 Type 26 (HASP Purge) Record [47]

Decimal Displacement	Field Size	Contents
0	1	Reserved (zero)
1	1	Record type (26)
2	4	Time record copied
6	4	Date record copied
10	2	System identification
12	2	System model identification
14	8	Job name
22	4	Reader start time
26	4	Reader start date
30	8	User identification
38	4	Reserved
42	2	Subsystem identification (2)
44	2	Section indicator
46	2	Descriptor section length
48	3	Reserved
51	1	Job information
52	4	HASP assigned job number
56	8	Job name
64	20	Programmer's name
84	1	Message class
85	1	Job class
86	2	Execution selection priority
88	2	Output selection priority
90	2	Input route code
92	8	Logical input device name
100	4	Programmer's account number
104	4	Programmer's box number
108	4	Estimated execution time
112	4	Estimated output lines
116	4	Estimated punched output
120	4	Default output form number
124	2	Print copy count
126	2	Lines per page
128	2	Print route code
130	2	Punch route code
132	2	Events section length
134	2	Reserved
136	4	Reader stop time
140	4	Reader stop date
144	16	Reserved
160	4	Execution start time
164	4	Execution start date
168	4	Execution stop time
172	4	Execution stop date
176	4	Output start time

Table C.3 continued

Decimal Displacement	Field Size	Contents
180	4	Output start date
184	4	Output stop time
188	4	Output stop date
192	2	Actuals section length
194	2	Reserved
196	4	Number of input cards
200	4	Generated output lines
204	4	Generated punched output
208	4	Reserved
212	4	Printed lines
216	4	Printed pages
220	4	Punched cards
224	2	Accounting identification
226	1	Job execution level
227	1	Local flags
228	2	Region in 64K units
230	1	Max disc requests in any step
231	1	Max tape 7 requests in any step
232	1	Max tape 9 requests in any step
233	1	Customer group data
234	4	Job selection priority
238	4	Accumulated customer time
242	4	Estimated I/O time
246	1	Print train mounts
247	1	Forms mounts
248	1	Accumulated tape mounts
249	1	Accumulated disc mounts
250	4	CPU time (.01 sec)
254	4	Charge calculation
258	4	I/O time (.01 sec)
262	4	Total pages in
266	4	Total pages out
270	1	Cancel rerun count
271	2	Cancel rerun explanations
273	4	CPU time lost on reruns
277	4	Memory charges lost on reruns
281	4	I/O time lost on reruns
285	16	Reserved

APPENDIX D

This appendix describes the two synthetic jobs designed as a part of this study. The first job was developed to emulate the resource demands of Autobatch cluster 4 (table 6.5(p.91)). The resource descriptor set used to characterize the demands of Autobatch jobs contained only three elements hence the synthetic job is quite simple. The second job was designed to emulate the resource demands of Batch cluster 6 (table 6.10(p.103)). The expanded resource descriptor set used to characterize the Batch jobs necessitates a more complex synthetic job.

The synthetic job designed for Autobatch cluster 4 is designed to allow the user to specify indirectly the number of lines printed and the total CPU time used by setting two parameters: NRLIN and NITER. The appropriate settings for these parameters may be determined using predictor equations established in section 6.7. A loop control parameter LIMIT = Maximum {NRLIN, NITER} is first calculated. The main loop is then executed a total of LIMIT times. The first NRLIN times through the loop, an output line is produced. Other actions accomplished each time through the loop include calculating two pseudo-random numbers using a multiplicative congruential scheme and performing some simple calculations on the second of these two generated numbers. The particular implementation of the job used in this study (WATFIV) is shown in figure D.1.

The synthetic job designed for Batch cluster 6 is somewhat more complex than the one designed for Autobatch cluster 4. Four parameters: NITER, NOUT, NTAP, and NDIS are specified to control the resource usage. NITER controls the number of times the "compute" loop is executed, NOUT controls how many lines of output are produced, NTAP controls how many records are read from a tape file, and NDIS controls how many records are read from a disk file.

The first task accomplished is to establish the loop control parameter LIMIT = Maximum {NITER, NOUT, NTAP, NDIS}. Within the main loop a pseudo-random number is produced. In addition, the first NOUT times through the loop a line is output; the first NTAP times through the loop a record is read from the tape file; the first NDIS times through the loop a record is read from the disk file; and the first NITER times through the loop a compute routine is invoked. The compute routine involves filling two 5x5 matrices with random numbers and then calling a routine to multiply the two matrices to form a third 5x5 product matrix. The appropriate settings for the parameters to produce a given demand pattern can be determined from predictor equations established in section 6.7. The particular implementation of the job used in this study (PL/I) is shown in figure D.2.

Fig. D.1 Program Listing for Autobatch Synthetic Job

```
C       SYNTHETIC JOB TO SIMULATE AUTOBATCH
        ISEED=21151
        NRLIN=0
        NITER=50
        LIMIT=NRLIN
        IF(NITER.GT.LIMIT)LIMIT=NITER
        DO 10 I=1,LIMIT
        CALL URANDX(ISEED,IRAND,URAND)
        IF(I.GT.NRLIN)GO TO 20
        WRITE(6,600)URAND,I
600     FORMAT(1X,'THE RANDOM NUMBER IS',1X,F7.5,1X,'ON ITERATION NR',
       *1X,I5)
20      CALL URANDX(ISEED,IRAND,URAND)
        ISEED=IRAND
        URAND=URAND+COS(URAND)*SIN(URAND)
10      CONTINUE
        STOP
        END
        SUBROUTINE URANDX(JSEED,IRAND,URAND)
C       THIS ROUTINE GENERATES A PSEUDO-RANDOM STANDARD UNIFORM
C       VARIATE STARTING WITH INTEGER SEED JSEED, AND RETURNING
C       A NEW SEED IRAND AND STANDARD UNIFORM VARIATE URAND.
        IRAND=JSEED*65539
        IF(IRAND)5,6,6
5       IRAND=IRAND+2147483647+1
6       URAND=IRAND
        URAND=URAND*.4656613E-9
        RETURN
        END
```

Fig. D.2 Program Listing for Batch Synthetic Job

```
SYNTH:PROCEDURE OPTIONS (MAIN);
/$THIS SYNTHETIC PROGRAM IS DESIGNED
TO EMULATE THE RESOURCE DEMAND
CHARACTERISTICS OF BATCH JOBS WHICH
PERFORM TAPE, DISK, AND UNIT RECORD
I/O. INPUT PARAMETERS ARE
  NITER - THE NUMBER OF TIMES THE
          COMPUTE LOOP IS TO BE EXECUTED,
  NOUT - THE DESIRED NUMBER OF
         OUTPUT LINES,
  NTAP - THE DESIRED NUMBER OF
         TAPE EXCPS TO BE ISSUED,
  NDIS - THE DESIRED NUMBER OF
        DISK EXCPS TO BE ISSUED,
  JSEED - AN INTEGER SEED USED TO
          BEGIN THE FIRST RANDOM
          NUMBER STREAM,
  ISEED - AN INTEGER SEED USED TO
          BEGIN THE SECOND RANDOM
          NUMBER STREAM.  $/
  ON FIXEDOVERFLOW ;
  ON ENDFILE (TAPIN) NTAP=0;
  ON ENDFILE (DISIN) NDIS=0;
/$DECLARE VARIABLES  $/
  DECLARE  SYSIN    FILE STREAM INPUT,
           SYSPRINT  FILE STREAM OUTPUT,
           TAPIN    FILE RECORD INPUT,
           DISIN    FILE RECORD INPUT,
           COMMON   BIT(10000) VARYING,
           JSEED    FIXED DEC(9),
           ISEED    FIXED DEC(9),
           UBAND    FLOAT DEC(6),
           IRAND    FIXED DEC(9),
```

Fig. D.2 continued

```
                LIMIT    FIXED DEC(9),
                NITER    FIXED DEC(9),
                NOUT     FIXED DEC(9),
                NTAP     FIXED DEC(9),
                NDIS     FIXED DEC(9),
                (I,J,K)  FIXED BINARY(31),
                (A(5,5),B(5,5),C(5,5))    FLOAT DEC(6);
/$ PREPARE FILES FOR PROCESSING  $/
  OPEN FILE (TAPIN) INPUT;
  OPEN FILE (DISIN) INPUT;
  OPEN FILE (SYSIN) INPUT;
  OPEN FILE (SYSPRINT) OUTPUT;
/$  READ INPUT PARAMETERS  $/
/$  COMPUTE CONTROL PARAMETERS  $/
  NITER=0;
  NOUT=0;
  NTAP=0;
  NDIS=0;
  JSEED=51121;
  ISEED=21151;
  LIMIT=NITER;
  IF NOUT>LIMIT THEN LIMIT=NOUT;
  IF NTAP>LIMIT THEN LIMIT=NTAP;
  IF NDIS>LIMIT THEN LIMIT=NDIS;
/$  COMPUTE LOOP  $/
MAINLOOP:DO I=1 TO LIMIT;
    CALL RAND(JSEED,IRAND,URAND);
    JSEED=IRAND;
/$  CHECK TO SEE IF OUTPUT LINE  $/
    IF I < NOUT THEN
       PUT SKIP LIST ('RANDOM NUMBER IS',
           URAND,'ON ITERATION NR',I);
/$  CHECK TO SEE IF READ TAPE  $/
```

Fig. D 2 continued

```
      IF I < NTAP THEN
         READ FILE (TAPIN) INTO (COMMON);
/$    CHECK TO SEE IF READ DISK  $/
      IF I < NDIS THEN
         READ FILE (DISIN) INTO (COMMON);
/$    CHECK TO SEE IF EXECUTE INNER LOOP  $/
      IF NITER < I THEN GO TO PASS_IT;
/$    INNER COMPUTE LOOP  $/
/$    THIS LOOP FILLS TWO 5X5 MATRICES
      A AND B WITH STANDARD UNIFORM
      RANDOM NUMBERS AND THEN INVOKES A
      ROUTINE TO MULTIPLY THE TWO MATRICES,
      RETURNING THE PRODUCT IN MATRIX C. $/
INLOOP:DO J=1 TO 5;
          DO K=1 TO 5;
             CALL RAND(ISEED,IRAND,URAND);
             ISEED=IRAND;
             A(J,K)=URAND;
             CALL RAND(ISEED,IRAND,URAND);
             ISEED=IRAND;
             B(J,K)=URAND;
           END;
  END INLOOP;
PASS_IT:;
 END MAINLOOP;
MATMUL:PROCEDURE(A,B,C);
  DECLARE   (A(5,5),B(5,5),C(5,5))  FLOAT DEC(6);
  DECLARE   (X,Y,Z)  FIXED BINARY(31);
/$    THIS ROUTINE MULTIPLIES TWO 5X5
      MATRICES A AND B AND PRODUCES THE PRODUCT
      MATRIX C.  $/
LOOP1:DO X=1 TO 5;
        DO Y=1 TO 5;
```

Fig. D.2 continued

```
            C(X,Y)=0.0;
          END;
  END LOOP1;
LOOP2:DO X=1 TO 5;
          DO Y=1 TO 5;
            DO Z=1 TO 5;
              C(X,Z)=A(X,Y)*B(Y,Z)+C(X,Z);
            END;
          END;
  END LOOP2;
  RETURN;
 END MATMUL;
RAND:PROCEDURE(JSEED,IRAND,URAND);
   DECLARE    (JSEED,IRAND)    FIXED DEC(9),
              URAND     FLOAT DEC(6);
/*  THIS ROUTINE GENERATES A PSEUDO-RANDOM
    STANDARD UNIFORM VARIATE URAND. THE STARTING
    SEED IS SUPPLIED THROUGH PARAMETER JSEED. A
    NEW SEED FOR THE NEXT PASS THROUGH THE
    PROCEDURE IS RETURNED THROUGH PARAMETER IRAND  */
  IRAND=JSEED*65539;
  IF IRAND < 0 THEN IRAND = IRAND+2147483647+1;
  URAND=IRAND;
  URAND=URAND*0.465661E-9;
  RETURN;
END RAND;
END SYNTH;
```

VITA

Wayne Thomas Graybeal was born to Stanley R. and Viola S. Graybeal on June 2, 1944 in Nucla, Colorado. He obtained the Bachelor of Science degree with a major in Mathematics from the University of Oklahoma in 1969. In 1970, he was awarded the Master of Arts degree with a major in Mathematics from the University of Arizona.

He enlisted in the United States Air Force in June, 1962, and has served continuously since that time. He was commissioned in May, 1969. His most recent assignments have been as a Space Object Identification Analyst, 13th Missile Warning Squadron, Clear AFS, Alaska (1970-71); Instructor, Course 30ZR2025B, USAFSAAS, Keesler AFB, Mississippi (1971-74); and Instructor/Assistant Professor, Department of Mathematical Sciences, U.S. Air Force Academy, Colorado (1974-1976).

Mr. Graybeal was married to Annie Elizabeth Pilcher on April 10, 1964. Three daughters (Shawn - 1965, Sandra - 1968, Susan - 1972) were born to this union.

Mr. Graybeal's permanent mailing address is:

P.O. Box 94

Nucla, Colorado 81424

The typist for this dissertation was Mrs. Annie E. Graybeal.


DATE
FILMED
12-84